Digi-Key CORPORATION
TechZone™ Magazine
TZM113.US

MICROCONTROLLER SOLUTIONS

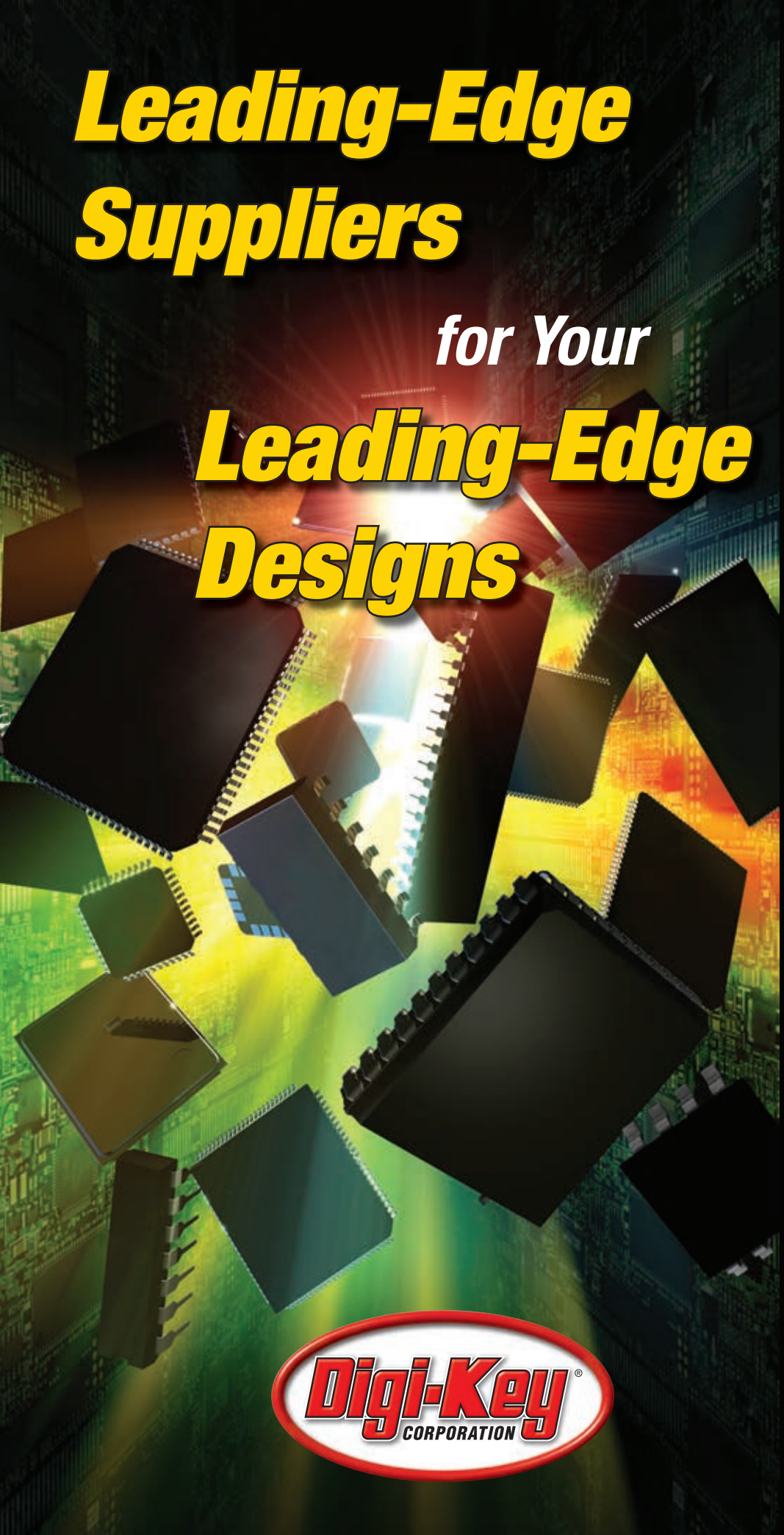
Look Inside Today's Microcontroller Technology

HUMAN INTERFACE

and the technology behind it

Understanding MCU Performance Analysis Techniques

Gain greater insight into your applications



ANALOG DEVICES

ATMEL

CIRRUS LOGIC

CYPRESS PERFORM

ENERGY micro

freescale semiconductor

FUJITSU

Infineon

MAXIM INNOVATION DELIVERED

MICROCHIP

NXP

PARALLAX

RENESAS

ROHM SEMICONDUCTOR

SILICON LABS

Texas Instruments

TOSHIBA Leading Innovation

zilog Embedded in Life An IXYS Company

Digi-Key CORPORATION TechZone™ Magazine

TZM113.US

MICROCONTROLLER SOLUTIONS

Digi-Key Features

TechXchange℠

Leading the Way in Collaboration

With any project, there will be obstacles to overcome, and answers to these questions are just a click away on Digi-Key's exciting new community forum. **16**

Understanding Microcontroller Performance Analysis Techniques

by Markus Levy, EEMBC

Processor benchmarks have minimized but not eliminated "specmanship," largely because of the limitations of older benchmarks. Understanding how the CoreMark benchmark works can provide useful insight into the trade-offs you may be inadvertently making with your MCU application. **18**

Table of Contents

ReferenceDesignLibrary

New Design Database Gives Digi-Key Customers a Competitive Edge

Digi-Key's Reference Design Library is among the best in online tools for electronic design engineers. **46**

Subscribe Today!

www.digikey.com/request

Editorial Comment

Mark Zack
Vice President, Semiconductors

As we witness the revolution of how we interact with the physical world, the necessity for enhanced tools and resources for human interface has become more evident. Despite the simplicity with which we view human senses, such as sight, sound, and touch, the technologies behind these interfaces are becoming increasingly complex.

By taking advantage of high-end microcontrollers, dedicated peripherals, and predefined software libraries, designers are able to implement their designs quickly with the latest technologies. As technology emerges and life cycles of products such as cell phones and media players continue to shrink, it is imperative to accelerate the design process.

In this issue of Digi-Key's *TechZone™ Magazine*, we offer a glimpse of the new and exciting solutions behind the human interfaces of tomorrow, with articles such as:

- Markus Levy of EEMBC reviews using the **CoreMark benchmark** as a reliable **indicator of performance** in "Understanding Microcontroller Performance Analysis Techniques" (page 18).
- Steve Gerber of Silicon Laboratories provides insight into advanced sensor-based interfaces for **gesture recognition** and **touch-less applications** in "Developing Next-Generation Human Interfaces Using Capacitive and Infrared Proximity Sensing" (page 37).
- Mitch Ferguson of Renesas Electronics America contributes an article on the **challenges and tradeoffs** associated with successfully implementing touch panel and touch key technology in "Minimize CPU Usage with Hardware-Assisted Touch Key Solutions" (page 22).

In addition to this magazine, visit **www.digikey.com/microcontrollers** to access a vast selection of development tools, application notes, Product Training Modules, and services. These resources can be used as a guide to device selection, education, and the development process, giving designers the confidence they need to complete their projects.

Recently, I reviewed a very interesting book by Frank Moss titled *The Sorcerers and Their Apprentices: How the Digital Magicians of the MIT Media Lab are Creating the Innovative Technologies That Will Transform Our Lives*. Moss is the former director of the MIT Media Lab, and in this book he chronicles how the development of future technologies will improve our quality of life. However, the most interesting take-away is the innovative methodology which MIT Media Lab employs to create break-through advancements in technology: bring multi-disciplined individuals together, throw out preconceived technology barriers, add in a dash of deadline and "hard fun", and the outcome is simply stunning. As a lead or member of a development team, you'll enjoy this read and the practical approach it can bring to your next design cycle.

To ensure that you receive notification of future editions of *TechZone™ Magazine*, visit **www.digikey.com/request**. Enjoy this issue and we look forward to your feedback!

Sincerely,

Mark Zack
Vice President, Semiconductors
Digi-Key Corporation

Digi-Key TechZone Magazine

About Digi-Key Corporation

As one of the world's leading, totally integrated global distributors of electronic components, Digi-Key Corporation has earned its reputation as an industry leader through its total commitment to service and performance. Digi-Key is a full-service provider of both prototype/design and production quantities of electronic components, offering more than two million products from over 470 quality name-brand manufacturers at www.digikey.com. A testament to Digi-Key's unparalleled commitment to service, North American design engineers have ranked Digi-Key #1 for Overall Performance for 19 consecutive years (EEITimes Distribution Study/October 2010). With global sales for 2010 surpassing $1.5 billion, Digi-Key's single location in North America is one of its greatest assets. Additional information and access to Digi-Key's broad product offering is available at www.digikey.com

Digi-Key's *TechZone™ Magazine* is a monthly series of technology-specific electronic publications featuring information and resources for Lighting, Microcontrollers, Wireless, and Sensors, with more technologies to come. *TechZone™ Magazine* provides the engineering community, from students to professional design engineers, with information about supplier innovations, quality in-depth solutions, and a selection of application-specific considerations focused on advancing technology. The archived and latest editions of *TechZone™ Magazine* can be found on Digi-Key's website.

Contact Information

For questions, comments, or to submit an article:
tzcontent@digikey.com

For information about advertising oppotunities:
techzoneadvertising@digikey.com

To be added to the email notification list:
www.digikey.com/request

Copyrights: The masthead, logo, design, articles, content and format of *TechZone* are Copyright 2011, Digi-Key Corporation. All rights are reserved. No portion of this publication may be reproduced in part or in whole without express permission, in writing, from Digi-Key.

Trademarks: DIGI-KEY, the Digi-Key logo, TECHZONE, and the TechZone logo are trademarks of Digi-Key Corporation. All other trademarks, service marks or product names are the property of their respective holders.

All product names, descriptions, specifications, prices and other information are subject to change without notice. While the information contained in this magazine is believed to be accurate, Digi-Key takes no responsibility for incorrect, false or misleading information, errors or omissions. Your use of the information in this magazine is at your own risk. Some portions of the magazine may offer information regarding a particular design or application of a product from a variety of sources; such information is intended only as a starting point for further investigation by you as to its suitability and availability for your particular circumstances and should not be relied upon in the absence of your own independent investigation and review. Everything in this magazine is provided to you "AS IS." Digi-Key expressly disclaims any express or implied warranty, including any warranty of fitness for a particular purpose or non-infringement. Digi-Key cannot guarantee and does not promise any specific results from use of any information contained in this magazine. Any comments may be submitted to techzone@digikey.com.

Microcontroller TechZone℠ Q & A

In the rapidly evolving microcontroller market, it seems as if there is something new every day – technologies, products, consumer trends, or regulations. Time-to-market demands have never been greater with the design of many of today's new microcontroller products requiring expertise in more than one discipline.

You have questions. We have answers. On target to field more than 260,000 calls this year, our technical support specialists are available 24/7/365 to answer your questions and assist you with your microcontroller needs.

If you have a question, we invite you to contact our technical staff via telephone, live web chat, or by emailing your question to **techzone@digikey.com**.

What are some of the benefits of Touch Sense buttons?

Touch Sense buttons are more reliable than their mechanical counterparts as there are no physical contacts to be worn out, resulting in a highly reduced mechanical failure rate. The designer will have a reduced bill of materials as no buttons have to be purchased. Touch Sense buttons are better suited for flat control panels. Additionally, Touch Sense buttons are flexible in design as there are multiple options available such as buttons, sliders, and dials.

Where can I find the Atmel QTouch Library?

The Atmel QTouch Library can be found at www.atmel.com. At the top of the Atmel homepage hover over the products tab and click on "Atmel QTouch Library" under Touch Solutions. The QTouch Library is royalty-free software. The user is required to register with Atmel prior to download.

What does it mean when an MP3 decoder is "bitstream compliant"?

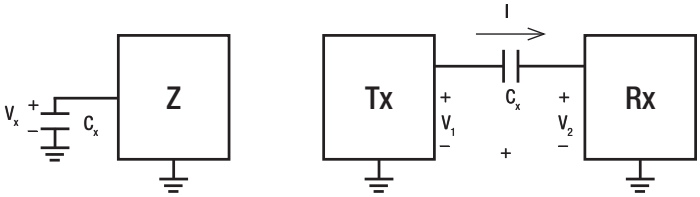
Being "bitstream compliant" means that the decompressed output from an MP3 file will be the same, within a specified degree of rounding tolerance. Decoders are compared, usually, by how computationally efficient they are (meaning how much memory or CPU time is used in the decoding process).

My design uses two microcontrollers communicating in a master/slave relationship using I²C. How do I keep the master from clocking out non-existent data from the slave if the slave is busy with a task?

Most microcontrollers provide for an I²C protocol feature called clock stretching. When the slave receives a read command and is busy, it will hold the clock line low. When the slave has placed data in the transmission register, it will release the clock line. Normally the clock line is low; the master brings the clock line high and then checks to see if it is actually high. If it is still low, the master understands that this indicates that the slave is holding the line low and should wait until it goes high before it proceeds. More can be found at www.i2c-bus.org.

What is the difference between self-capacitance and mutual capacitance?

Self-capacitance uses a single pin to measure capacitance between that pin and ground. It operates by driving current on a pin connected to a sensor and measures the voltage. An example of self-capacitance is that when a finger touches the sensor, the capacitance increases. Applications for self-capacitance include single touch sensors like buttons and sliders. Mutual capacitance, on the other hand, uses a pair of pins to measure capacitance. Mutual capacitance operates by driving current on a transmit pin and measures the charge on a receive pin. An example of this is when a finger is placed between the transmit (Tx) and receive (Rx) pins, measured capacitance decreases. Applications for mutual capacitance includes multitouch systems like touchscreens and trackpads. (See diagram below)



Self-Capacitance Mutual Capacitance

Do you have a question about microcontroller solutions?

Digi-Key has more than 130 technical support specialists, product managers, and applications engineers who are eager to answer your questions and assist you with your microcontroller projects.

Send your questions to **techzone@digikey.com**.

Secure Microcontrollers Keep Data Safe

by Dave Bursky, *PRN Engineering Services*

Secure MCU offerings range from 8-bit to 32-bit CPUs with dedicated encryption engines, random number generators, and additional features to secure communication channels and protected data.

In this internet age, identity theft, intellectual property protection, and financial account and payment protection are key concerns to both consumers and designers. To keep everything safe, many systems employ security measures such as data encryption and physical shielding to prevent hackers and other malicious activities from accessing data, financial information, or even intellectual property. Even the simple car door entry key/ignition key has become more secure with embedded processors running challenge-and-response authentication to prevent vehicle theft. Furthermore, the movement to "smarten" the energy grid will also escalate the demand for secure communications to prevent hackers or terrorists from wreaking havoc on the power grid.

Although general-purpose embedded processors can do the encryption and decryption of the data, the compute-intensive requirements of encryption standards such as DES (data encryption standard), AES (advanced encryption standard), Elliptic Curve, SHA-1 (secure hash algorithm), and others can bog down a processor and slow down the overall transaction. To speed up the computations on its advanced processors, Intel added new instructions to accelerate AES encryption and decryption, as well as AES-Galois Counter Mode (AES-GCM) authenticated encryption. When running at multi-GHz speeds, these processors can deliver performance comparable to dedicated encryption solutions.

However, power consumption for a desktop-class Intel i3, i5, i7 or Xeon-class processor could hit 50 W or more. Such a solution is not practical for embedded systems that run constantly, have limited cooling capabilities, and require milliwatt power consumption levels.

In past years, to speed up the computations, separate data-encryption chips have been implemented, but the connection between the processor and the encryption chip is a good target for hackers. To prevent hackers from accessing the connections between chips, designers enclosed the board in a metal shield and included tamper sensors that erased the data and encryption keys if any tampering was detected.

Today, the level of integration possible allows processor vendors to integrate the encryption engine into their embedded processor/microcontroller, thus making the system a bit more secure. The dedicated engine accelerates the computations so that transactions can be done in real-time with no noticeable delay, thus reducing the wait for the user and allowing the system to handle more transactions per minute.

Many embedded processor vendors – Atmel, Freescale, Maxim, Microchip, NXP, PalmChip, STMicroelectronics, Texas Instruments, and others – have included dedicated encryption/decryption engines and random number generators on their processor chips. Additionally, encryption engine blocks are available from several vendors of intellectual property, and such blocks can be co-integrated with a processor core on a custom chip, or embedded in a field-programmable gate array along with a processor core.

In the financial market, there are several key standards for equipment such as banking and credit card terminals. One such standard, PCI PTS 3.0, is the latest effort of the Payment Card Industry Security Standards Council (PCI SSC), which was created by many of the payment-products companies – MasterCard, VISA, American Express, and JCB. This standard deals with the logical and physical approaches that attempt to extract security personal identification number (PIN) codes and encryption keys from point-of-interaction (POI) systems such as banking terminals (automated teller machines and credit card terminals) and other systems.

Processor choices range from 8 to 32 bits

Depending on the application requirements, designers have a wide range of encryption solutions, from low-cost, 8-bit microcontrollers to top-performing, 32-bit embedded processors. At the high end of the spectrum are several security processors with dedicated on-chip encryption engines based on an ARM CPU core – the AT91SAM family from Atmel, the ZA9L1 from Maxim, the ST33 series from STMicroelectronics, and the CC430 series from Texas Instruments, to name a few. Additional secure processors based on other 32-bit cores include multiple families based on the ColdFire and PowerPC cores offered by Freescale Semiconductor, and a family of secure chips based on the SmartMX2 CPU core from NXP.

The AT91SAM family, for example, uses an ARM7TDMI Thumb 32-bit processor and includes 128 to 512 kb of flash code storage and from 32 to 128 kb of high-speed SRAM. Both AES and triple DES encryption engines are included on the chip. The AES engine handles 256-, 192-,

or 128-bit key algorithms and is compliant with the FIPS PUB (Federal Information Processing Standard Publication) 197 specifications. The triple DES engine handles two-key or three-key algorithms and is compliant with the FIPS PUB 46-3 specifications. A full system-on-a-chip, the AT91SAM processors also include many other system resources – a USB port, a 10/100 Ethernet MAC, a CAN controller, multiple serial ports, multiple timer/counter blocks, an 8-channel 10-bit A/D converter, and other system support functions.

Moving up to an ARM922T 32-bit processor, the ZA9L1 Zatara processor is also a highly integrated system-on-a-chip which runs at up to 200 MHz (see Figure 1). Supporting the processor are multiple tamper sensor inputs, an AES 128-bit encryption/decryption engine, a true random number generator for key and challenge creation, a secure boot mechanism to ensure code authenticity, and 4 kb of zeroizing (tamper detection will set all bits to zero to erase the memory contents if the circuitry detects an intrusion), non-volatile, static RAM for secret storage. The chip has the horsepower and the security features to tackle sensitive applications which place high demands on system performance. Thanks to the secure boot capability, designers have the flexibility of using off-chip storage for the control programs and are not limited by on-chip flash storage, such as that used by many of the other secure processors.

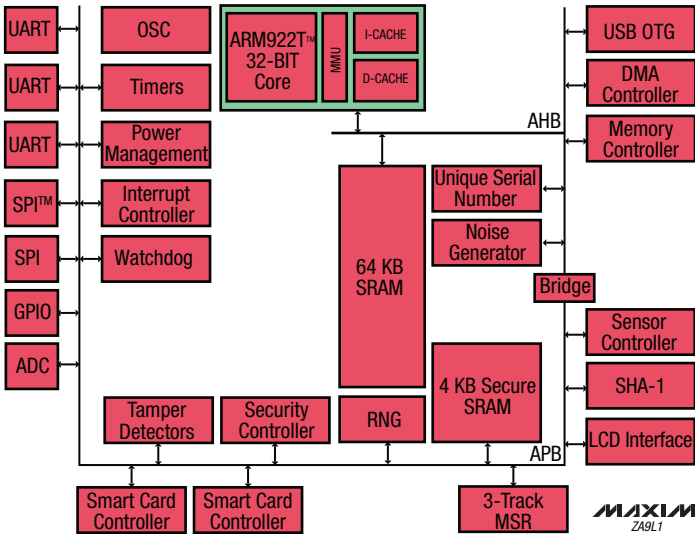


Figure 1: Maxim's ZA9L1 Zatara secure processor. (Source: Maxim Integrated Products. Used with permission.)

Offering the largest on-chip flash storage, the ST33 series developed by STMicroelectronics packs up to 1.25 Mb of flash as well as up to 30 kb of RAM. Based on an ARM SC300 core, the ST33 series includes the company's NESCRYPT (Next Step Cryptography) cryptographic engine for public-key cryptography, a true random number generator, and a DES accelerator. Each chip also includes a unique serial number and an ISO 3309 CRC (cyclic redundancy code) calculation block that can be used to help in detecting program or data tampering. Targeting at applications such as smart cards, mobile TV, and banking applications, the chip can operate from power supplies as low as 1.8 V.

The NESCRYPT engine supports BAC (basic access control), EAC (extended access control), and AA (active authentication).This platform implements very fast e-passport transactions (in less than three seconds), and also supports the IAS ECC specification based on the European Citizen Card (ECC). It has been certified by Common Criteria EAL6+ (Evalutation Assurance Level). STMicroelectronics claims that the ST33 series is the world's first secure microcontroller series to achieve EAL6+ certification, according to the Common Criteria 3.1 methodology.

Security offerings from Texas Instruments include multiple families of products based on different ARM cores – the Stellaris family employs an ARM Cortex-M3, both the Sitara and Integra families use the ARM9 and Cortex-A8, and the DaVinci digital media processors use the Cortex-A8. The AM3894/3892 media processors run the Cortex-A8 core at 1.2 GHz and include AES, Triple DES, and a random number generator on the chip along with a three-dimensional graphics engine and high-definition video encoding/decoding.

8- and 16-bit processors are up to the task, too

Taking aim at applications such as smart card readers, USB secure tokens, and financial terminals, the MAXQ1050 and MAXQ1850 are based on 32-bit, internally developed RISC processor cores and include accelerators which perform high-speed encryption with AES, RSA, DSA, ECDSA, SHA-1, SHA-224, SHA-256, DES, and triple DES algorithms. The chips also include an Random Number Generator for key generation and challenge generation. Also incorporated on the chip is a sophisticated security mechanism to protect secret key data when the processor is under attack. Two self-destruct inputs and environmental sensors (temperature and voltage) can be set to erase secret key data when an attack is detected.

Other non-ARM based, 32-bit secure processors are available from vendors such as Freescale (using either the ColdFire or PowerPC processor cores) and from NXP, which employs its SMARTX2, a proprietary 32-bit processor core. The offerings from Freescale include the recently released QorIQ quad-core processor based on the PowerPC which targets mixed control-plane and data-plane applications. The on-chip SEC 4.2 encryption engine handles many algorithms – public key acceleration, DES, AES, message digest accelerator, random number generation, ARC4, SNOW 3G F8 and F9, CRC, and Kasumi.

Introduced by NXP in late 2010, its new IntegralSecurity architecture was designed to protect the integrity and confidentiality of user data and applications targeting CC EAL 6+ certification. IntegralSecurity is based on over 100 dedicated security mechanisms which create a dense protection shield including redundancy and multiple layers. A hardened Fame2 crypto coprocessor, also developed by NXP, provides even more DPA resilience, serving the full range of RSA/ECC crypto algorithms with a flexible RSA key length of up to 4,096 bits. In addition, the SmartMX2 includes the NXP-patented SecureFetch feature, which protects against light and laser attacks, and also covers data other than software code. Lastly, the processor also includes NXP's patented GlueLogic feature for advanced protection against reverse-engineering attacks.

When the high throughput of a 32-bit processor is not needed, designers can select from many low-power 8- and 16-bit microcontroller solutions. Freescale, Inside Secure, Maxim, Microchip, NXP, PalmChip, and STMicroelectronics are just some of the suppliers of microcontrollers with embedded encryption engines. The SecureAVR series offered by Inside Secure (previously offered by

Atmel), for example, includes a proprietary 8/16-bit RISC processor core and hardware DES, Triple DES, and hardware AES engines that are all DPA/DEMA resistant. Also on board is a checksum accelerator, both 16- and 32-bit CRC engines, and a 32-bit AdvX cryptographic accelerator for public key operations which comes with a cryptographic library (RSA, ECC, key generation, and other functions).

One of the chips in the family, the AT90SO128, (see Figure 2) uses the Atmel SecureAVR RISC processor core, which allows the linear addressing of up to 8 Mb of code and up to 16 Mb of data as well as a number of new functional and security features. The AdvX cryptographic engine, developed by Atmel and licensed to Inside Secure, is a 32-bit accelerator dedicated to performing fast encryption and authentication functions. It is combined with a 32 kb ROM that stores the secure crypto firmware. The ability to map on-chip EEPROM into the code space allows parts of the program memory to be reprogrammed in-system. This technology combined with the versatile 8/16-bit CPU provides a highly flexible and cost-effective solution to many smart-card applications. Additional security features include power and frequency protection logic, logical scrambling on program data and addresses, power analysis countermeasures, and memory accesses controlled by a supervisor mode. The chip also includes dedicated hardware for protection against SPA/DPA/SEMA/DEMA attacks, as well as protection against physical attacks including an active shield.

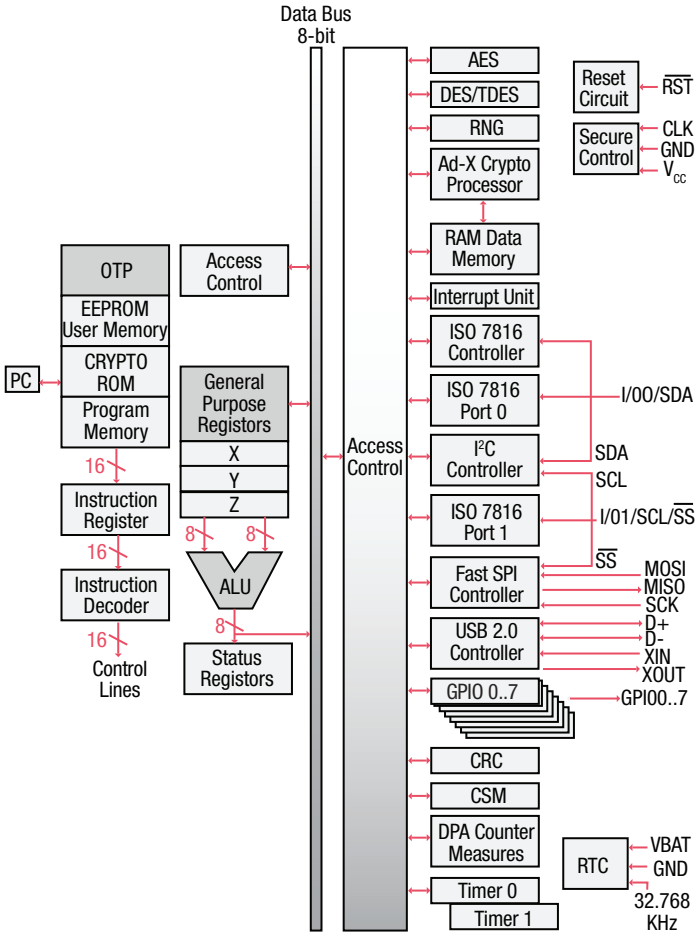


Figure 2: Based on a proprietary 8/16-bit SecureAVR processor core, the AT90SO128 offered by Inside Secure includes hardware DES, and Triple DES and AES engines. (Source: Inside Secure. Used with permission.)

Offering a range of secure 8- and 16-bit solutions, Maxim's MAXQ series of low-power microcontrollers include several devices that leverage the company's 16-bit MAXQ pipelined RISC CPU. The MAXQ1004, for example, contains an AES encryption engine, a random number generator, and the company's proprietary 1-Wire slave interface (see Figure 3). Another device, the MAXQ1010, includes both a DES accelerator and an AES engine, allowing applications to rapidly respond to challenges and authenticate other devices by using standards-based cryptography. A true random-number generator is also on the chip and it can be used for key generation, random padding, challenge generation, and other applications. This processor also contains a 128-byte secure key storage memory which is instantly erased when a "self-destruct" input is received. Both the MAXQ1004 and the 1010 have ultra-low-power stop modes which cut the current drain to less than 400 nA (typically), helping conserve power in battery-operated systems.

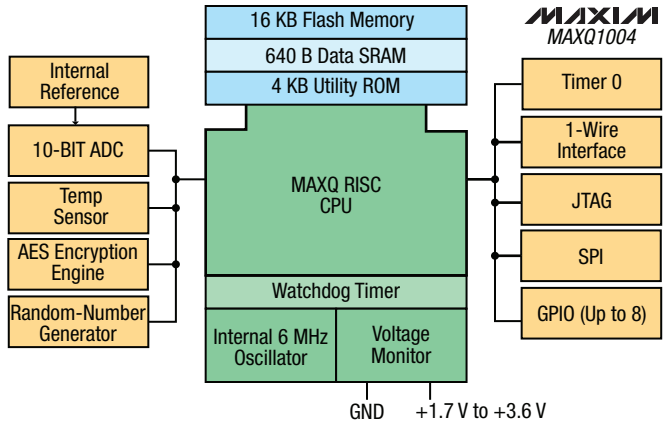


Figure 3: A 16-bit MAXQ pipelined RISC CPU developed by Maxim is at the heart of the MAXQ1004, which the company targets for portable electronics, battery chargers, and battery packs. (Source: Maxim Integrated Products. Used with permission.)

Oldies but goodies

Even the venerable 8-bit 8051 is still a viable microcontroller when co-integrated with a crypto accelerator (see Figure 4). The AcurX51 secure microcontroller from PalmChip is a good example of leveraging a low-cost controller core for markets such as the smart grid and home area networks. The revamped 8051 core executes instructions in a single cycle, thus considerably improving the execution efficiency of the basic instruction set. Coupling that with a dedicated encryption engine, the processor delivers enough performance for most smart-grid applications.

Also employing an 8051 at their hearts, the DS5002 and DS5003 from Maxim store data in encrypted form using an on-chip 64-bit key. Included in both chips is a true Random Number Generator to create the key, and a self-destruct input to blank the memory should the chip detect tampering. Another family member, the DS5250, targets applications such as PIN pads, financial terminals, and other security applications. The chip encrypts its program memory and can also optionally encrypt its data memory with a hardware-based, single- or triple-DES algorithm, making it almost impossible to extract information. The chip also employs block cipher encoding that uses block addresses to modify the encrypted data, still further enhancing the data security.

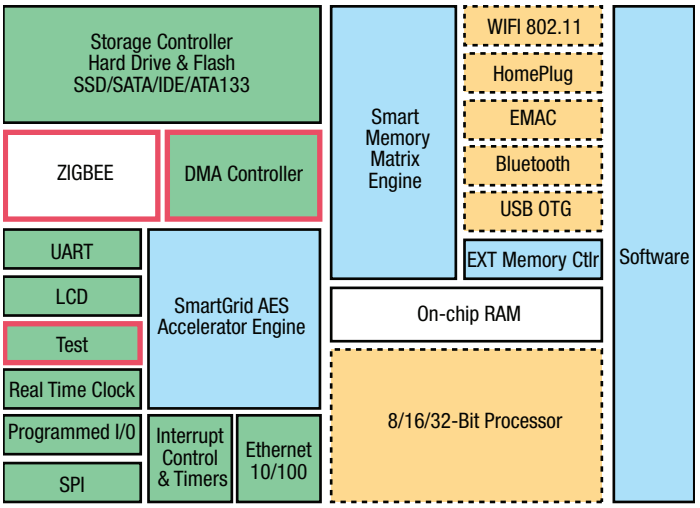


Figure 4: A typical system-on-a-chip based on the Acurx51 modular architecture can employ an 8-, 16-, or 32-bit RISC processor, depending on the expected workload. (Source: PalmChip. Used with permission.)

Based on its proprietary 8-bit PIC processor core, Microchip also offers security-enhanced processors, such as the PIC12F635/PIC16F636/639. These 8-bit processors include a cryptographic module the company calls KEELOQ (see Figure 5). The module employs a block-cipher encryption algorithm based on a block length of 32 bits and a key length of 64 bits. The algorithm obscures the information in such a way that even if the unencrypted/challenge information differs by only one bit from the information in the previous challenge, the next coded response will be totally different. Statistically, if only one bit in the 32-bit string of information changes, approximately 50 percent of the coded transmission will change.

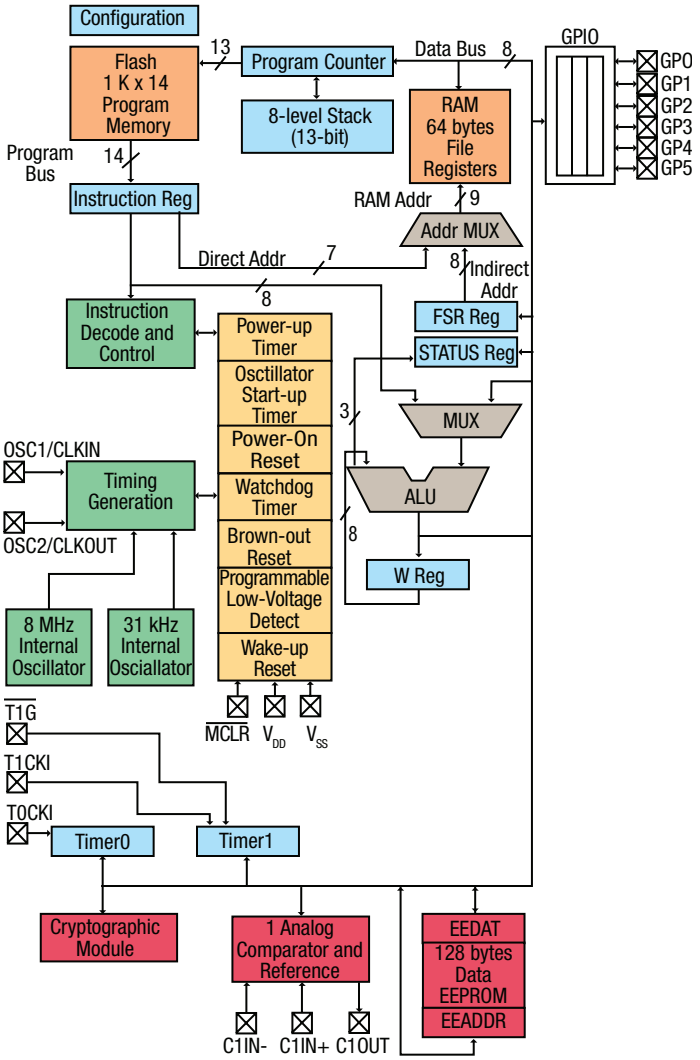


Figure 5: The PIC12F636 is built around Microchip's 8-bit PIC processor core and incorporates the company's KEELOQ cryptographic module that employs a block-cipher algorithm to encrypt the data. (Source: Microchip. Used with permission.)

Atmel's AT91SAM7X/XC Flash-based Microcontrollers

Atmel

Atmel's SAM7X and XC MCUs are highly integrated, flash-based devices based on the ARM7TDMI core. Main features include the integrated 10/100 Ethernet MAC, a peripheral DMA controller, and more.

This Product Training Module (PTM) provides self-paced learning instruction, introduces engineers to Atmel's ARM7-based innovations, and allows users to learn which tools, evaluation kits, and real-time operating systems are available for this family.

PTM PRODUCT TRAINING MODULES *Online...On Demand*®

www.digikey.com/ptm


SUPPORT FOR
ENGINEERS
Design Support Services Team
• Provides in-depth application assistance and advice on system design
• Assists with product selection and development tools
• Produces application notes, webinars, and instructional videos
• DSS website: www.digikey.com/design
• Contact: design@digikey.com
Technical Support Team
• Available 24/7 via telephone, email, and live web chat
• Provides customers with product-specific information, cross-reference, and component recommendations
• Contact: techs@digikey.com
Digi-Key®
CORPORATION
Providing customers with access to in-depth design assistance and product-specific information, as well as detailed specifications and performance on new products.
The industry's broadest product selection available for immediate delivery
www.digikey.com
1.800.344.4539

Ethernet Throughput on NXP ARM Microcontrollers

by Sergio Scaglia and Kenneth Dwyer, *NXP Semiconductors*

This article presents a method for measuring Ethernet throughput, providing a good estimate of performance, and illustrating the different factors that affect performance.

Ethernet is the most widely installed Local Area Network (LAN) technology in the world. It has been in use since the early 1980s and is covered by the IEEE Std 802.3, which specifies a number of speed grades. In embedded systems, the most commonly used format runs at both 10 Mbps and 100 Mbps (and is often referred to as 10/100 Ethernet).

There are more than 20 NXP ARM MCUs with built-in Ethernet, covering all three generations of ARM (ARM7, ARM9, and the Cortex-M3). NXP uses essentially the same implementation across three generations, so designers can save time and resources by reusing their Ethernet function when systems move to the next generation of ARM.

This article discusses three different scenarios for measuring Ethernet throughput on the LPC1700 product and details what is really achievable in an optimized system.

Superior implementation

NXP's Ethernet block (see Figure 1) contains a full-featured 10/100 Ethernet MAC (media access controller) which uses DMA hardware acceleration to increase performance. The MAC is fully compliant with IEEE Std 802.3 and interfaces with an off-chip Ethernet PHY (physical layer) using the Media Independent Interface (MII) or Reduced MII (RMII) protocol along with the on-chip MII Management (MIIM) serial bus.

The NXP Ethernet block is distinguished by the following:

- Full Ethernet functionality – The block supports full Ethernet operation, as specified in the 802.3 standard.
- Enhanced architecture – NXP has enhanced the architecture with several additional features including receive filtering, automatic collision back-off and frame retransmission, and power management via clock switching.
- DMA hardware acceleration – The block has two DMA managers, one each for transmit and receive. Automatic frame transmission and reception with Scatter-Gather DMA offloads the CPU even further.

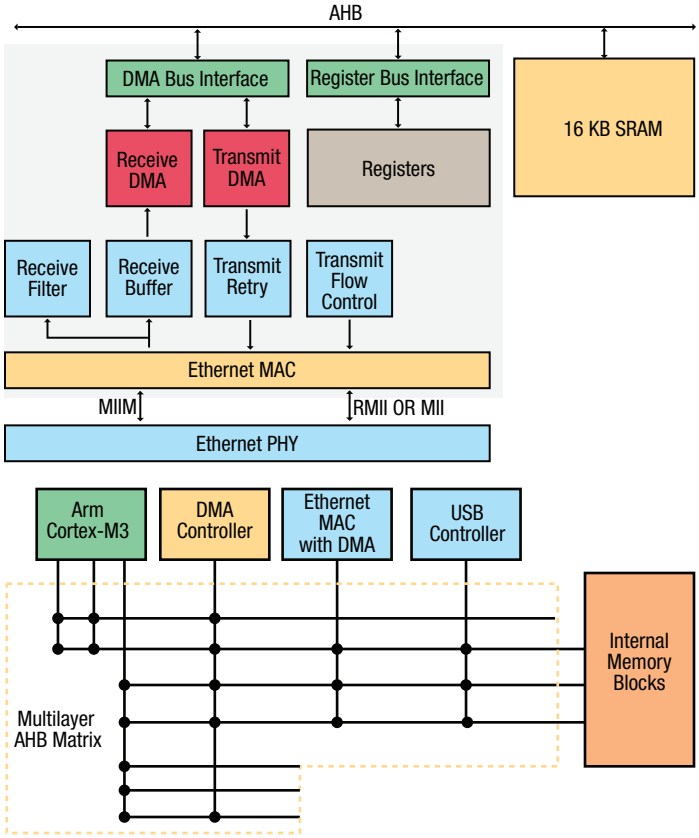


Figure 1: LPC24xx Ethernet block diagram. NXP's Cortex-M3 architecture.

Ethernet throughput on NXP's LPC1700 microcontrollers

In an Ethernet network, two or more stations send and receive data through a shared channel (a medium), using the Ethernet protocol. Ethernet performance can mean different things for each of the network's elements (channel or stations). Bandwidth, throughput, and latency are measures which contribute to overall performance. In the case of the channel, while the bandwidth is a measure of the capacity of the link, the throughput is the rate at which usable data can be sent over the channel. In the case of the stations, Ethernet performance can mean the ability of that equipment to operate at the full bit and frame rate of the Ethernet channel. On the other hand, latency measures the delay in time caused by several factors (such as propagation times, processing times, faults, and retries).

The focus of this article will be on the ability of the NXP LPC1700 to operate at the full bit and frame rate of the Ethernet channel to which it is connected, via the Ethernet interface (provided by the internal EMAC module plus the external PHY chip). In this way, throughput will be defined as a measure of usable data (payload) per second, which the MCU is able to send/receive to/from the communication channel. The same concepts can also be applied to other NXP LPC microcontrollers supporting Ethernet.

Unfortunately, these kinds of tests generally require specific equipment such as network analyzers and/or network traffic generators, in order to get precise measurements. Nevertheless, using simple test setups it is possible to get estimated numbers. In fact, our goal is to understand the different factors that can affect Ethernet throughput, so users can focus on different techniques in order to improve Ethernet performance.

Here only the throughput of the transmitter is considered, as the case of the receiver is a little bit more complex because its performance is relative to the performance of the transmitter that put the information into the channel. In this case, the throughput of the receiver will be affected by the throughput of the transmitter sending the data over the channel. Once we get the throughput for the transmitter, we can consider this number as the maximum ideal number that the receiver will be able to achieve (under ideal conditions), and get the throughput for the receiver relative to this number.

Reference information

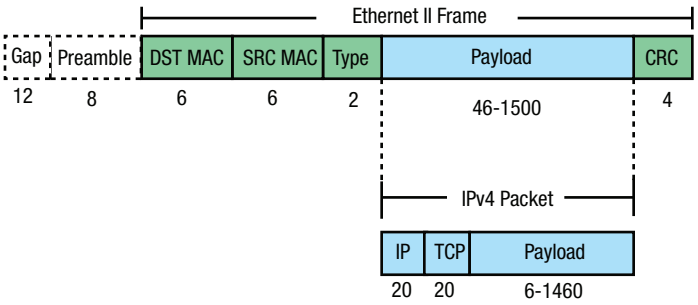


Figure 2: Ethernet II frame.

Considering a bit rate of 100 Mbps, and that every frame consists of the payload (useful data, minimum 46 bytes and maximum 1,500 bytes), the Ethernet header (14 bytes), the CRC (4 bytes), the preamble (8 bytes), and the inter-packet gap (12 bytes), then the following are the maximum possible frames per second and throughout:

For minimum-sized frames:
(46 bytes of data) -> 148,809 frames/sec -> 6.84 Mb/sec

For maximum-sized frames:
(1,500 bytes of data) -> 8,127 frames/sec -> 12.19 Mb/sec

The above rates are the maximum possible values which are in reality impossible to reach. Those values are ideal and any practical implementation will have lower values (see Figure 2).

Notes:

- Frames/second is calculated by dividing the Ethernet link speed (100 Mbps) by the total frame size in bits (84 * 8 = 672 for minimum-sized frames, and 1,538 * 8 = 12,304 for maximum-sized frames).
- Megabytes/second is calculated by multiplying the frames/second by the number of bytes of useful data in each frame (46 bytes for minimum-sized frames, and 1,500 bytes for maximum-sized frames).

Test conditions (see Figure 3)
MCU: LPC1768 running at 100 MHz
Board: Keil MCB1700
PHY chip: National DP83848 (RMII interface)
Tool chain: Keil µVision4 v4.1
Code running from RAM
TxDescriptorNumber = 3
Ethernet mode: Full duplex – 100 Mbps

Test description
In order to get the maximum throughput, there are 50 frames consisting of 1,514 bytes (including Ethernet header), each consisting 75 Kb of payload (useful data). The CRC (4 bytes) is automatically added by the EMAC controller (Ethernet controller).

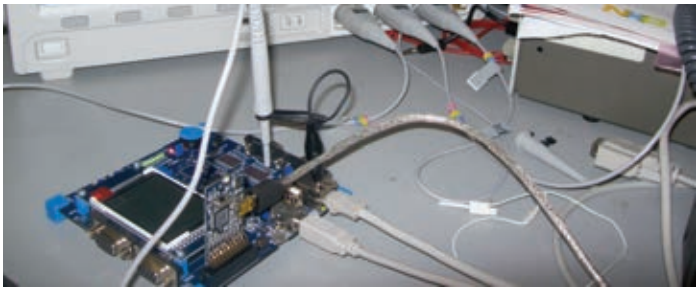


Figure 3: The test setup.

In order to measure the time this process takes, a GPIO pin is set (P0.0 in our case) just before starting to send the frames and is cleared as soon as we finish with the process. In this way, an oscilloscope can be used to measure the time as the width of the generated pulse at the P0.0 pin. The board is connected to a PC using an Ethernet cross cable.

The PC runs a sniffer program (WireShark in this case, http://www.wireshark.org/) as a way to verify whether the 50 frames were sent and the data is correct. A specific pattern in the payload is used so any errors can be easily recognized. If the 50 frames arrive at the PC with no errors, the test is considered valid (see Figure 4).

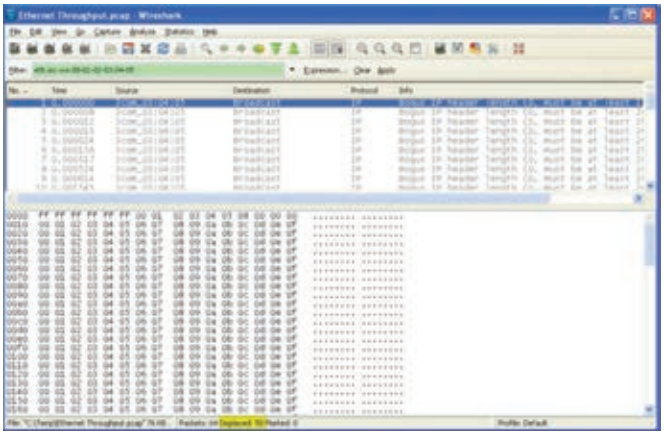


Figure 4: Verifying the payload.

Test scenarios

The EMAC uses a series of descriptors which provide pointers to memory positions where the data buffers, control, and status information reside. In the case of transmission, the frame data should be placed by the application into these data buffers. The EMAC uses DMA to get the user's data and fill the frame's payload before transmission. Therefore, the method the application uses in order to copy the application data into those data buffers will affect the overall measurement of the throughput. For this reason, three different scenarios are presented:

1. An "ideal" scenario, which doesn't consider the application at all,
2. A "typical" scenario, where the application copies the application's data into the EMAC's data buffers, using the processor,
3. An "optimized" scenario, where the application copies the application's data into the EMAC's data buffers, via DMA.

Scenarios description

1. **"Ideal" scenario:** In this case, the software sets up the descriptors' data buffers with the test's pattern, and only the TxProduceIndex is incremented 50 times (once for every packet to send) in order to trigger the frame transmission. In other words, the application is not considered at all. Even though this is not a typical user's case, it will provide the maximum possible throughput in transmission.
2. **"Typical" scenario:** This case represents the typical case in which the application will copy the data into the descriptors' data buffers before sending the frame. Comparing the results of this case with the previous one, it is apparent that the application is affecting the overall performance. This case should not be considered as the actual EMAC throughput. However, it is presented here to illustrate how non-optimized applications may lower overall results giving the impression that the hardware is too slow.
3. **"Optimized" scenario:** This test uses DMA in order to copy the application's data into the descriptors' data buffers. This case considers a real application but using optimized methods which effectively take advantage of the fast LPC1700 hardware.

Software

Test software in the form of a Keil MDK project is provided for this article (please check NXP's website for AN11053).
The desired scenarios can be selected by using the Configuration Wizard and opening the "config.h" file (see Figure 5). Besides the scenario, both the number of packets to send and the frame size can also be modified through this file.

Test results

After running the tests, the following results are tabulated as demonstrated in Table 1:

Table 1: Test results.

	Frames Sent	Payload (bytes)	Total Data (bytes)	Time (msec)	Throughput (Mbytes/sec)	% relative to Max. Possible
Max Possible					12.19	100.0%
Scenario 1	50	1500	75000	6.25	12.00	98.44%
Scenario 2	50	1500	75000	10.44	7.18	58.93%
Scenario 3	50	1500	75000	7.1	10.56	86.66%

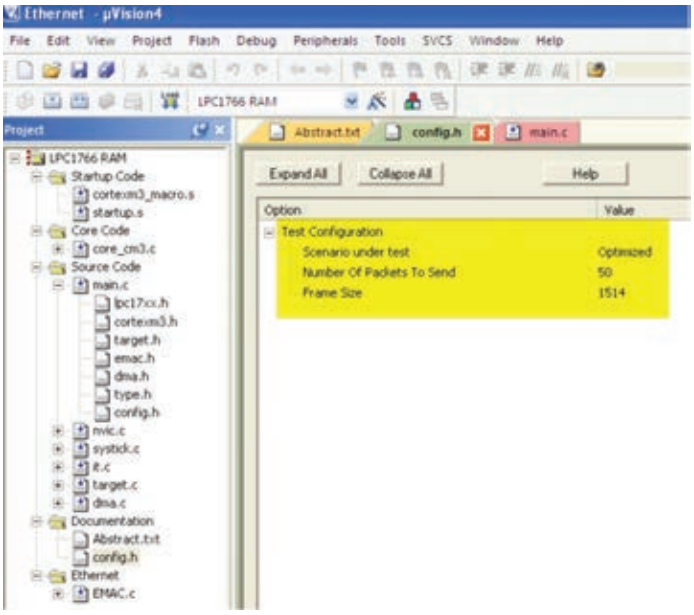


Figure 5: Choosing the test scenarios.

Conclusion

Despite the fact that Scenario 1 is not a practical case, it provides the maximum value possible for our hardware as a reference, which is very close to the maximum possible for Ethernet at 100 Mbps. In Scenario 2, the application's effects on the overall performance become apparent. Finally, Scenario 3 shows how an optimized application greatly improves the overall throughput.

Other ways to optimize the application and get better results were found by running the code from flash (instead of from RAM), and in some cases by increasing the number of descriptors.

In summary, Ethernet throughput is mainly affected by how the application transfers data from the application buffer to the descriptors' data buffers. Improving this process will enhance overall Ethernet performance. The LPC1700 and other LPC parts have this optimization built in to the system hardware with DMA support, enhanced EMAC hardware, and smart memory bus architecture.

NXP's LPC1700 Series

NXP Semiconductors

The LPC1700 series is based on the ARM Cortex-M3 core, revision two.
The core offers several enhancements over the ARM7 core, including advanced math features like single cycle multiply and hardware divide.

NXP PTM PRODUCT TRAINING MODULES *Online...On Demand*®

www.digikey.com/ptm

DesignServicesProviders

Digi-Key's Design Services Providers network offers fee-based, high-quality, efficient, system-level engineering and consulting services, including design feasibility, hardware/software development, prototyping, PCB design, and manufacturing. Consulting services provided include ARM Cortex, FPGA, WiFi, ZigBee, Linux, DSP, and lighting technologies. With expertise in a wide range of product applications and embedded systems, these design firms can help you get to market faster, finish your project on time, and provide short-term assistance.

www.digikey.com/designservices

LPC177x/8x

Industry's first Cortex™-M3 MCU shipping with integrated LCD controller

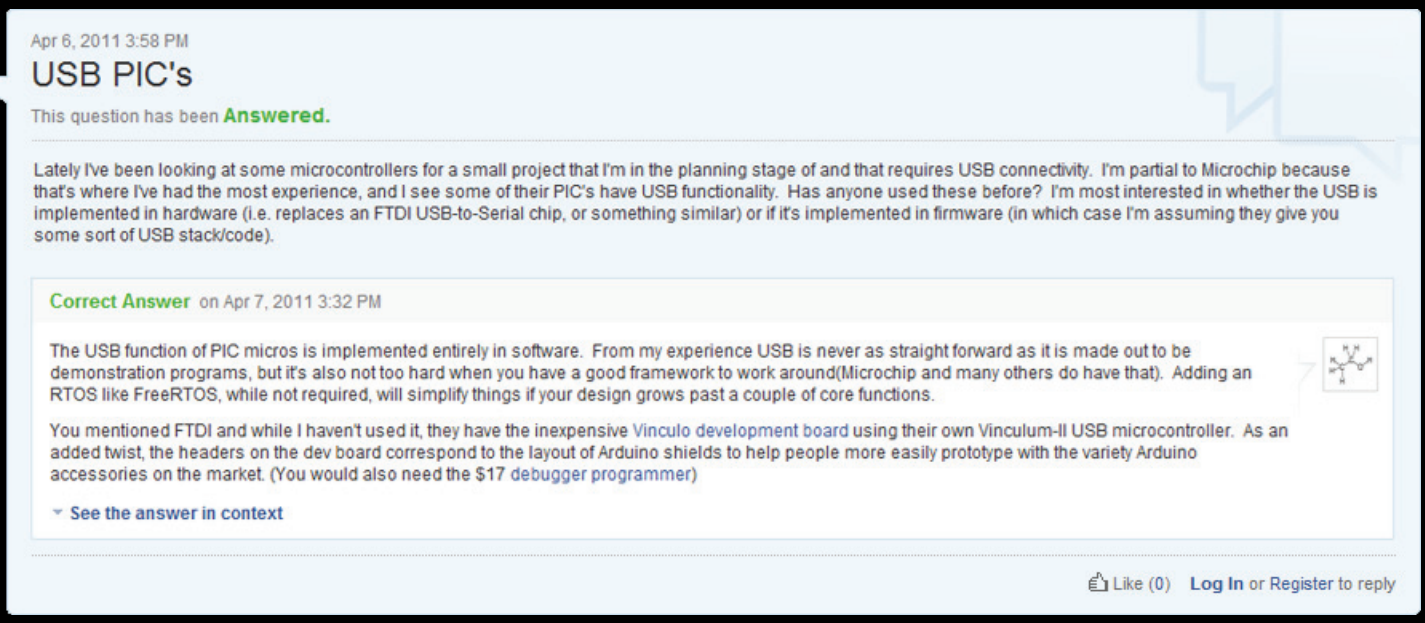
- LCD interface includes DMA controller with built-in FIFO to support STN & TFT displays up to 1024x768 pixels
- 96 KB SRAM and 32-bit external memory interface
- Up to 512 KB Flash, 4 KB EEPROM, USB, Ethernet, and CAN 2.0B
- Supported by industry leading hardware/software tools

www.digikey.com/nxp-mcu

TechXchange℠

Leading the Way in Collaboration

Whether you are a home hobbyist repairing a television, an engineering student constructing a robot for a final project, or an engineer for an electronics OEM inventing the latest gadget, problems will arise. With any project, there will be obstacles to overcome, and answers to these questions and concerns are not always easy to acquire. The Internet is a fantastic resource for a myriad of opinions and possibilities, but sifting through the inadequacies to find credible sources is not always easy. The answers you need are just a click away on Digi-Key's exciting new community forum.



Enter TechXchange℠ *beta*, Digi-Key's newest addition to an already impressive list of features available, at no cost to you, on its award-winning website, **www.digikey.com**. TechXchange was born of Digi-Key's continuing initiative to provide its customers with the best service possible, from design to implementation. Help with technical problems has always been just a phone call away, as Digi-Key's talented staff of technical advisors are always ready and willing to help customers with any issues that may arise. Now you can access solutions from the convenience of your home computer through the forum on TechXchange, the next step in improving customer access to technical assistance and allowing individuals to have a larger role in their own problem-solving process.

TechXchange is a community forum where everyone from design engineers to hobbyists can discuss technology, products, designs, and more. Users can participate in discussions, ask questions of the community, and learn about new products and the latest innovations. The forum is divided into five separate sections, or communities: Lighting, Microcontroller, Power, Sensors, and Wireless. A General Community is also available if your question does not fit easily into a specific category. Each section is moderated by approved subject matter specialists who are ready and able to answer any questions that are posted in the forum. Users are also encouraged to share their expertise and reply to posted questions.

Each of the TechXchange sections coincides with a corresponding TechZone℠ on the Digi-key website. These TechZones are invaluable resources that go hand-in-hand with TechXchange by providing product guides, application notes, reference designs, white papers, product training modules and more for the over two million products that are available on the Digi-Key website from over 470 electronic component suppliers. Users also have direct access to *TechZone™ Magazine* via the link at the top of the section with which the magazine corresponds.

TechXchange includes a fully-searchable discussion archive. Every discussion that appears on TechXchange will be available via search. The value of this feature will only grow as the content of the site increases day by day.

To ensure users are getting quality answers from reliable sources, Digi-Key has implemented a rating system for TechXchange. This system indicates those individuals, whether Digi-Key staff or not, who have the highest level of accuracy and participation within the community. These people are given special recognition for their commitment in a special box on each section page.

As TechXchange grows, so will the diversity of opinions that are given through the community experience, making it a valuable resource for important information. So, when that seemingly insurmountable problem arises during your next project, remember that Digi-Key is always here to help, and that TechXchange should be your first stop in your quest for information.

Become a member today!

www.digikey.com/techxchange

Understanding Microcontroller Performance Analysis Techniques

by Markus Levy, *EEMBC*

Benchmarks let you compare processors, but there is still plenty of variability. Understanding and running standard benchmarks can give designers greater insight into and control over their applications.

It has never been easy to compare microprocessors. Even comparing desktop or laptop computers, which typically have processors that are all variants of the same fundamental architecture, can be frustrating as the one with faster numbers may run achingly slow compared to a "worse" one. Things get even tougher in the embedded world, where the number of processors and configurations is practically infinite.

Benchmarking is the usual solution to this conundrum. For many years, the Dhrystone benchmark (a play on the Whetstone benchmark, which includes the floating-point operations which Dhrystone omits) was the only game in town. However, it has a number of significant issues. Chief among them is the fact that it doesn't reflect any real-world computation, it simply tries to mimic the statistical frequency of various operations. In addition, compilers can often do much of the computing at compile time, meaning the work doesn't have to be done when the benchmark is run.

The real test of a benchmark is that, when looking in detail at results (especially ones that initially look strange), you can rationalize why the results look the way they do. An ideal benchmark would provide a score that purely reflected the processor's performance capabilities, irrespective of the rest of the system. Unfortunately, that's not possible because no processor acts in isolation: all processors must interact with memory – cache, data memory, and instruction memory, each of which may or may not run at the full processor frequency.

In addition, these processors must all run code generated by a compiler, and different compilers generate different code. Even the same compiler will generate different code depending on the optimization settings chosen when you compile your code. Such differences cannot be avoided, but the main thing to avoid is having the actual benchmark code optimized away.

Even though results might depend on the compiler and memory, you should be able to explain any such results only in terms of the processor itself, the compiler (and settings), and memory speed. That was not the case with the Dhrystone benchmark. However, the more recent CoreMark benchmark from the Embedded Microprocessor Benchmark Consortium (EEMBC) has overcome these deficiencies and proven to be much more successful.

EEMBC developed and publicly released the CoreMark benchmark in 2009 (downloaded now by more than 4,100 users). It has been adapted for numerous platforms, including the Android. The developers took specific care to avoid the pitfalls of older benchmarks. By looking at how the CoreMark benchmark works, as well as some example results, we can see that it not only can be a reliable indicator of performance, but it can also help to identify where microcontroller and compiler performance can be improved.

The CoreMark benchmark program

The CoreMark benchmark program uses three basic data structures to represent real-world work. The first structure is the linked list, which exercises pointer operations. The second is the matrix; matrix operations typically involve tight optimized loops. Finally, state machines require branching that is hard to predict and which is much less structured than the loops used for matrix operations.

In order to remain accessible to as many embedded systems as possible, big and small, the program has a 2 kb code footprint.

Figure 1 illustrates how the program works. The first two steps might seem trivial, but they are actually critically important – they are the steps that ensure the compiler can't pre-compute any results. The input data to be used are not known until run time.

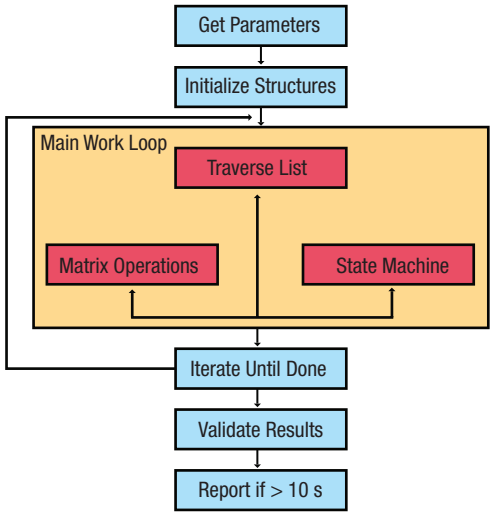


Figure 1: CoreMark benchmark process.

The bulk of the benchmark workload happens in the main work loop. One of the data structures, a linked list, is scanned. The value of each entry determines whether a matrix operation or a state-machine operation is to be performed. This decision-operation step is repeated until the list is exhausted. At that point, a single iteration is complete. The work loop is repeated until at least 10 seconds have elapsed. The 10-second requirement is imposed to ensure enough data to provide a meaningful result. If it runs for less than this, then the benchmark program will not report a result. However, this time requirement can be modified if the user runs the benchmark on a simulator.

After the main work is completed, the common cyclic redundancy check (CRC) function is utilized; which acts as a self-check to ensure that nothing went wrong (by accident or otherwise) during execution. Assuming everything checks out, the program reports the CoreMark result. This number represents the number of iterations of the main work loop per second of execution time.

While most benchmark users are honest, it's always important to ensure that any benchmarking scheme has protections against abuse. There are two primary ways that someone could try to tamper with the results: editing the code (other than within the porting layer), since the code is necessarily provided in source form, and simply faking results. The CRC helps to detect any issues that might arise if the code is corrupted, and certification by the EEMBC Technology Center is the final arbiter. No one is required to have their results certified, but certification adds significant credibility since it confirms that a neutral third party achieved the same numbers.

Adapting the benchmarking runs

While the program looks to user parameters for initializing the data, you do not explicitly provide the raw data. That would be too much work, and it would also open up the possibility of manipulation through careful selection of initialization data. Instead, the program looks for three-seed values which must be set by the user. These numbers direct the initialization of values in the data structures in a way which is opaque to the user. While they act as "seeds," there is no random element to this. The structures are completely deterministic, and multiple runs of a benchmark with the same seeds will result in identical execution and results.

You may also need to adapt the benchmark to account for the way in which your system allocates memory. Systems with ample resources can simply use the heap and `malloc()` calls. This allows each allocation of memory to be made when needed and for exactly the amount of memory needed. That flexibility comes at a cost, however, and better systems need faster ways of using memory.

The fastest approach is to pre-allocate memory entirely, but with linked list operations that is not feasible. An intermediate approach is to create a number of predefined chunks of memory (a memory pool) which can be doled out as needed. The tradeoff is that you cannot select how much memory you get in each chunk – you get a fixed size block.

The porting layer allows you to adapt the benchmark to the type of memory allocation scheme used on the system being evaluated.

Parallelism is another trait that you can exploit if your system supports it. You can build the CoreMark benchmark for parallel operation, specifying the number of contexts (threads or processes) to be spawned during execution. However, you should avoid using CoreMark for representing a processor's multicore performance since this benchmark will most certainly scale 99.9 percent linearly due to its small size.

Making sense of the results

Of course, it is what you do with the benchmark results that can lead to confusion (intentional or otherwise). For this reason, EEMBC has imposed strict reporting requirements. The CoreMark website, http://www.coremark.org, has a place for reporting results, and you can not just enter the one CoreMark score. There are a few other critical variables that may affect your results.

- The biggest is the compiler used and the options set when compiling the benchmark. You must report that information when submitting results.
- A second major influencer is the means by which memory is allocated – you must report the approach taken if it is something other than heap (`malloc`).
- A third factor, when relevant, is parallelization. You must report the number of contexts created.

There is also an alternative way you can report the results. Rather than specifying the raw CoreMark value, you can normalize the clock frequency in order to focus on the architectural efficiency by itself. This is a CoreMark/MHz value which measures the number of iterations per million clock cycles. If you report this number, then you must also report the memory speed relative to the processor clock speed. If the cache frequency can be configured relative to the processor frequency, that configuration must also be reported.

A look at some specific results can help show how the various required reporting elements correlate with different benchmark scores, and why it's important to identify these elements when reporting your results. All of the numbers that follow come from the public list of results available on the CoreMark website.

The simple impact of compiler version can be seen in the results of Table 1. The Analog Devices processor shows a speedup of ten percent with a newer compiler version, presumably indicating that the new compiler is doing a better job. The Microchip example shows an even greater difference between two more distant versions of the GNU C compiler (gcc). For the two NXP processors, all of the compilers are subtle variations on the same version thereby minimizing the differences.

Table 1: Impact of different compilers on CoreMark results.

Processor	Compiler	CoreMark/MHz
Analog Devices BF536	gcc 4.1.2	1.01
	gcc 4.3.3	1.12
Microchip PIC32MX36F512L	gcc 3.4.4 MPLAB C32 v1.00-20071024	1.90
	gcc 4.3.2 (Sourcery G++ Lite 4.3-81)	2.30
NXP LPC1114	Keil ARMcc v4.0.0.524	1.06
	gcc 4.3.3 (Code Red)	0.98
NXP LPC1768	ARM CC 4.0	1.75
	Keil ARMCC v4.0.0.524	1.76

Even when using the same compiler, different settings will, of course, create different results as the compiler tries to optimize the program differently. Table 2 shows some example results.

Table 2: Changing compiler settings yields different CoreMark results.

Processor	Compiler	Settings	CoreMark/MHz
Microchip PIC24FJ64GA004	gcc 4.0.3 dsPIC030, Microchip v3_20	-Os -mpa	1.01
		-03	1.12
Microchip PIC24HJ128GP202	gcc 4.0.3 dsPIC030, Microchip v3.12	-03	1.86
		-mpa	1.29
Microchip PIC32MX36F512L	gcc 4.4.4 MPLAB C32 v1.00-20071024	-02	1.71
		-03	1.90

In the first example (PIC24FJ64GA004), optimizing for smaller code size comes at a cost of reduced benchmark performance of about 10 percent. The difference is even more dramatic in the second case, running 30 percent slower when the procedural abstraction optimization flag (-mpa) is set. The difference between the compiler settings on the last processor also reflects differences in the amount of optimization, with approximately a 10 percent benefit from the greater speed optimizations provided by the –O3 setting.

The effects of memory can be seen in Table 3. In the first case, wait states are introduced when the clock frequency exceeds what the flash can handle, reducing the CoreMark/MHz number. Similarly, in the second case, the DRAM can't keep up with the processor above 50 MHz, so the clock frequency ratio between the memory and processor goes down to 1:2, reducing the CoreMark/MHz number.

Table 3: Effect of memory settings on CoreMark results.

Processor	Clock Speed	Memory Notes	CoreMarck/ MHz	CoreMark
TI OMAP 3530	500	Code in FLASH	2.42	1210
	600		2.19	1314
TI Stellaris LM3S9B96 Cortex M3	50	1:1 Memory/CPU clock not possible beyond 50 Mhz	1.92	96
	80		1.60	128

However, in both of these cases, the increase in clock frequency was greater than the decrease in operational efficiency, so the raw CoreMark number still went up with the increase in clock frequency; it just didn't increase as much as the frequency did.

Finally, the impact of cache size can be seen in Table 4. Here, the code fits in the 2 kb cache of the first configuration, but it fills the cache completely. None of the function parameters on the stack can fit, and so there will be some cache misses. In the second case, the cache has twice the capacity, meaning that it doesn't suffer the same cache misses as in the first example, giving it a better score.

Table 4: Effect of changing the cache size on CoreMark results.

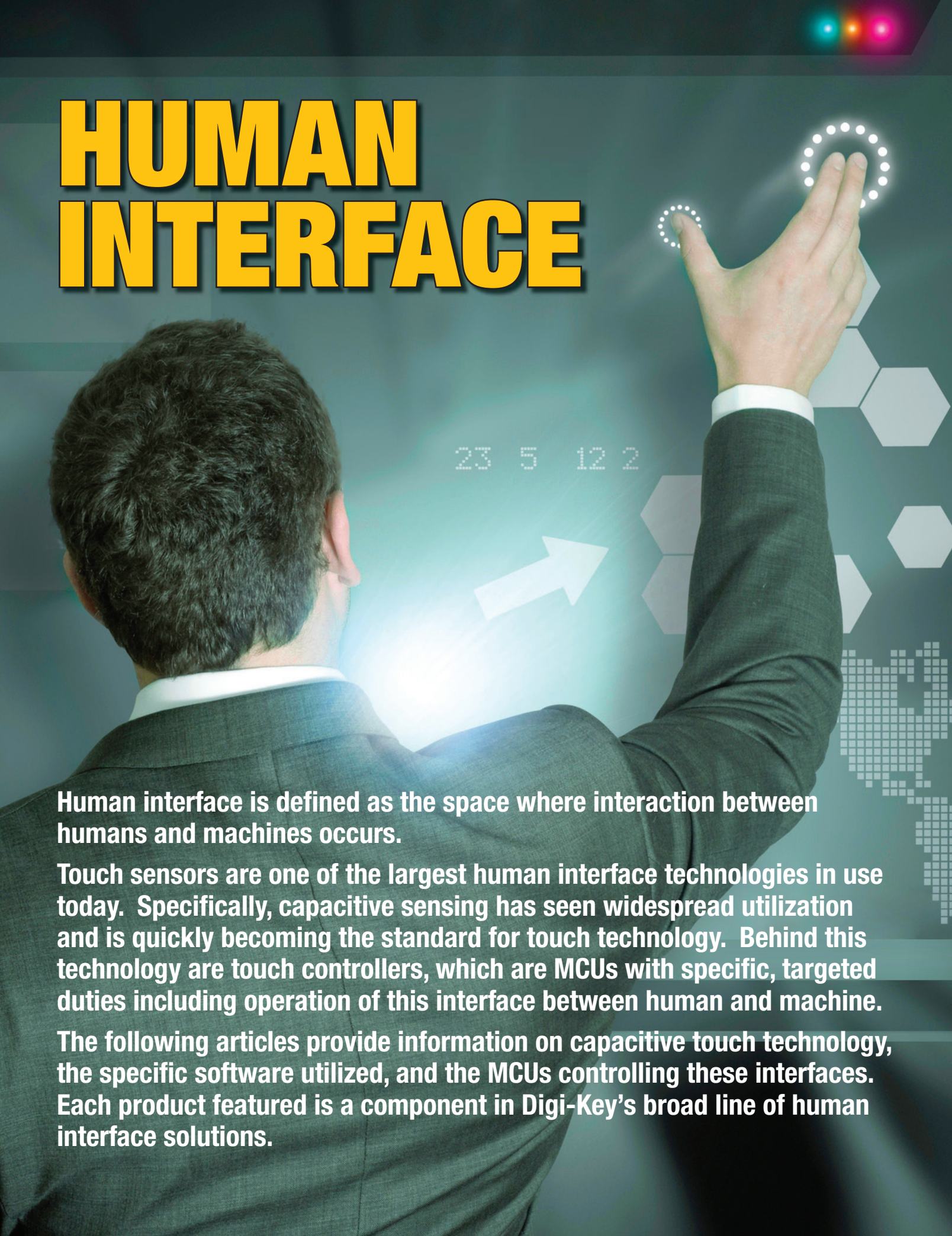
Processor	Cache Size	CoreMark/MHz
Xilinx MicroBlaze v7.20.d in Spartan xC3S700A, 3-stage	2 kb	1.48
Xilinx MicroBlaze v7.20.d in Spartan xC3S700A, 5-stage	4 kb	1.66

Note that the second case has a five-stage pipeline, compared to the three-stage pipeline of the first case. A longer pipeline should cause performance to go down due to the extensive branching of the state machine example. A longer pipeline takes longer to refill when a branch is mis-predicted. So the higher CoreMark score indicates that the larger cache more than made up for this degradation.

The scores in all of these examples demonstrate two facts. First, a single number (in this case, CoreMark/MHz) can accurately represent the performance of the underlying microcontroller architecture; the latter examples bear that out clearly. Second, however, is the fact that context matters. The compiler can influence how well the code it generates performs. This should almost be obvious – people spend a lot of time developing and refining compilers to improve the results they create, but "good" results depend on whether your goal is fast code or small code. Faster code will run faster, smaller code will not (except in those cases where it actually helps optimize the memory or cache utilization).

The most important thing that these examples show, however, is that the differences between results have rational explanations. They're not caused by some figment of the actual benchmarking code that might favor one microcontroller architecture over another, or by the outright dropping of code by the compiler. In that sense, the CoreMark benchmark is fair and balanced, giving a true reflection of the compiler and architecture, and only of the compiler and architecture.

HUMAN INTERFACE



Human interface is defined as the space where interaction between humans and machines occurs.

Touch sensors are one of the largest human interface technologies in use today. Specifically, capacitive sensing has seen widespread utilization and is quickly becoming the standard for touch technology. Behind this technology are touch controllers, which are MCUs with specific, targeted duties including operation of this interface between human and machine.

The following articles provide information on capacitive touch technology, the specific software utilized, and the MCUs controlling these interfaces. Each product featured is a component in Digi-Key's broad line of human interface solutions.

Minimize CPU Usage with Hardware-Assisted Touch Key Solutions

by Mitch Ferguson, *Renesas Electronics America*

Touch-panel and touch-key technology design and implementation are on the rise. Touch panels offer many advantages for embedded-system products, ranging from the ergonomic to the aesthetic, but there are several challenges and trade-offs that must be considered in order to successfully implement these technologies.

Making the optimum design choice is easier when one is familiar with the key technologies, understands the challenges, and follows design guidelines which aid the system development process. While there are several microcontroller (MCU)-based capacitive touch measurement methods currently on the market, hardware-assisted solutions offer engineers the most ideal touch implementation approach to help them overcome the challenges associated with integrating embedded touch technologies.

Integrated hardware-based implementation

Integrating MCU with capacitive touch sensing offers several advantages including:

- Single-chip solutions
- Reduced CPU usage for touch function
- Minimal system resource requirements
- Reduced development cycles
- Reduced power consumption

Implementing the touch-key functions in hardware saves a significant number of CPU cycles, which can then be used to implement system control. Additional features can also be added to improve the amount of CPU bandwidth available for system control management.

To support design engineers entrusted with the development of human-machine interfaces, leading MCU supplier Renesas Electronics has developed an integrated solution that combines a 16-bit R8C CPU core with a Touch Sensor Control Unit (T-SCU).

R8C/3xT capacitive touch key solution

The R8C/3xT group of MCUs incorporates a dedicated hardware block called the Sensor Control Unit (SCU) to perform touch sensing while maintaining minimum CPU usage, which helps to significantly reduce power consumption levels compared to traditional solutions. The SCU also provides full programmability to automate the touch detection process and integrates mechanisms that allow for improved noise tolerance.

Sensor control unit

The SCU provides sensing during standby mode and supports up to four electrodes per channel. The SCU handles four key functions: Control and error management, automated scanning and measurement, noise counter measures, and data transfer.

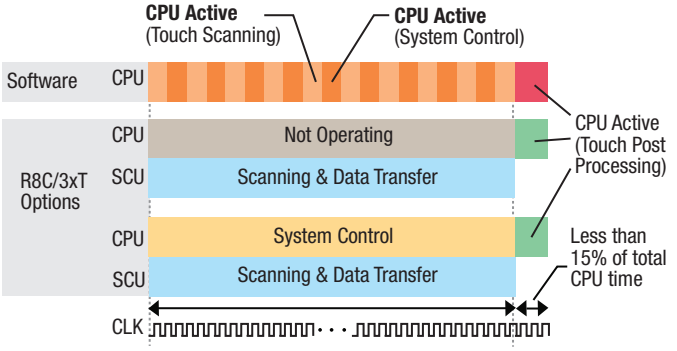


Figure 1: Over 85 percent of CPU bandwidth is available.

Control and error management

As illustrated below, the SCU is comprised of a status counter, secondary counter, and primary counter. The SCU controls the ports, the counters, and data transfer to detect the floating capacitance of the capacitive touch electrode.

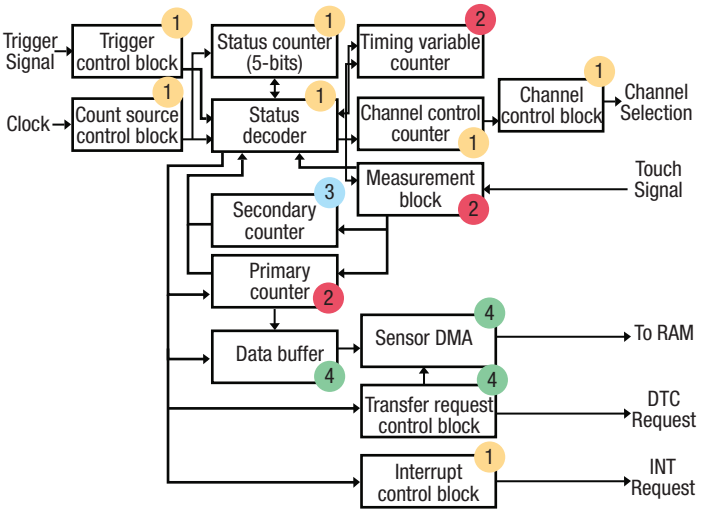


Figure 2: T-SCU block diagram.

Automated scanning and measurement

The SCU manages the automated scanning, freeing the CPU to focus on system control functions. The SCU features two modes of operation:

- Single Mode – single-channel touch detection
- Scan Mode – multiple-channel touch detection, either sequentially or selectively. The scanning can be triggered either in software, using an RC timer or an external trigger

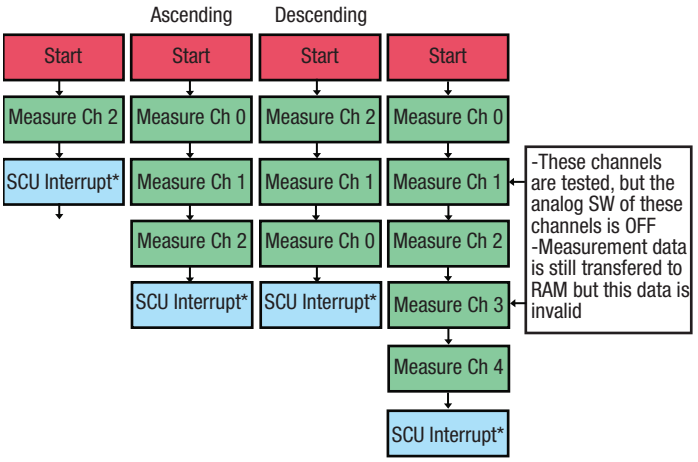


Figure 3: Automatic scanning offlaods the CPU.

Noise counter measures and environmental variations

The SCU has the ability to filter out noise from the touch measurement systems, implementing either low-frequency or RF noise cancellation, resulting in accurate touch decisions.

Table 1: Noise counter measures.

Type	Frequency Band	Noise Source	Filtering Technique	Hardware/ Software
Switching	1 kHz- 1 MHz	-Inductive heating noise -Magnetic field noise -Power supply -Dimming noise	-Secondary counter method (low frequency noise cancellation)	Hardware (SCU)
			-Additional averaging process	Software
RF	100 kHz- 900 MHz	-AM wave noise	-Multiple measurement techniques	Hardware (SCU)
Environmental variations	< 1 kHz	-Temperature changes -Characteristic drift over time -Stray capacitance	-Drift correction processing	Software

Low-frequency noise cancellation:

The secondary counter sets the number of measurements after the voltage falls below the detection threshold. The SCU can then increment the secondary counter in the event that a detection threshold crossing occurs before the counter goes down to zero, thereby rejecting any kind of spike variations.

PATTERN A			PATTERN B		
port	n	m	port	n	m
H	95	7	H	95	7
H	96	7	H	96	7
H	97	7	H	97	7
H	98	7	H	98	7
H	99	7	H	99	7
L	100	7	L	100	7
L	100	6	H	101	8
L	100	5	H	102	9
L	100	4	L	102	8
L	100	3	L	103	7
L	100	2	H	103	8
L	100	1	L	103	7
L	100	0	L	103	6
			L	103	5
			L	103	4
			L	103	3
			L	103	2
			L	103	1
			L	103	0

Step 1

Step 2

n-Value of primary counter
m-Value of secondary counter
L-Higher than the detect threshold voltage
H-Lower than the detect threshold voltage

Figure 4: The secondary counter is used to eliminate low-frequency noise.

RF noise cancellation:

The SCU implements multiple methods to eliminate RF interference including random measurements, measurement by majority decision and a combination of both.

- ***Random measurement:*** The SCU hardware can randomly vary the sample point of each sensor to minimize the detection effects of radiated and conducted noise sources. This hardware-based method has the advantage of obtaining the desired measurements while minimizing the CPU usage. Users have 16 different timing options to select from, which helps in noise rejection during touch measurement.

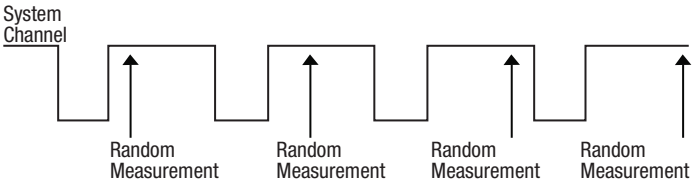


Figure 5: One of 16 random sampling points can be used for the measurement.

- ***Measurement by majority decision:*** This method measures the number of times set during the measurement period and judges "H"/"L" from measurement results using decision by majority.

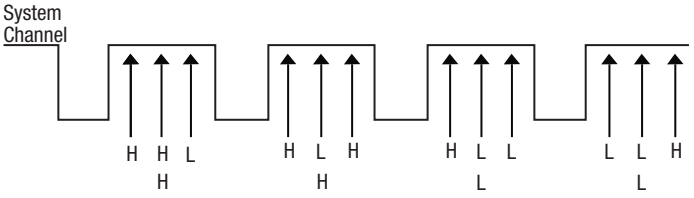


Figure 6: This approach filters out high-frequency noise.

Data transfer

The SCU can also manage the transfer of the measured values to a RAM buffer set up in the memory.

In a selective scan mode, the RAM buffer will contain data for all channels from the start channel to max channel, even if the enable bit for a channel is not set.

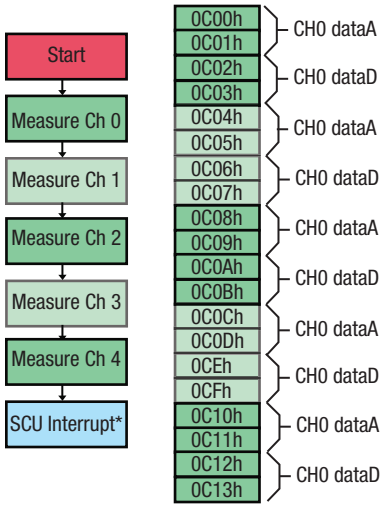
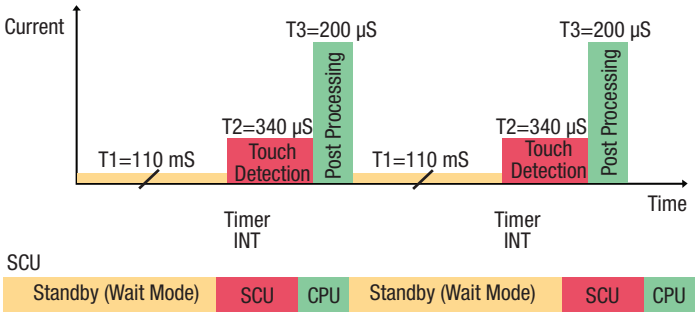


Figure 7: The DTC helps transfer the data without CPU intervention.

Lower power consumption

The SCU's touch sensing capability in standby mode also helps minimize the average current draw, for example, by approximately 16 µA in a typical response time cycle of 100 ms.



Clock Source	Low-speed OCO	High-speed OCO	High-speed OCO
Peripheral Clock	125 kHz	5 MHz	5 MHz
CPU State	Stop	Stop	Active
Code Execution	N/A	N/A	N/A
State Current*	8.3 µA	610 µA (single ch)	2 mA
Average Current	Sub 16 µA (single channel)		

Figure 8: Sensing in Wait Mode helps reduce the overall average power consumption.

Software Architecture

As shown in Figure 9, the Renesas Electronics touch solution consists of four layers.

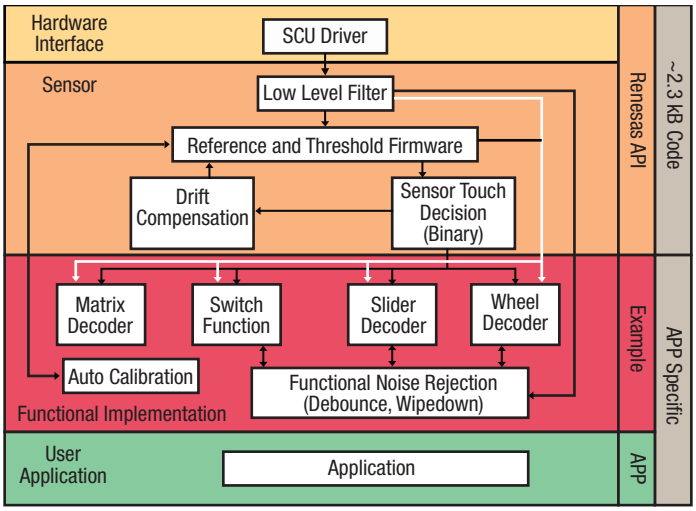


Figure 9: The Touch API is only ~1.2 KB in size.

- Hardware Interface Layer – contains the low-level drivers that help configure the SCU block
- Sensor Layer – handles the processing and makes touch decisions. It also contains:
  - Drift-compensation routines
  - Noise countermeasures (e.g., low-level filtering)
  - Touch Decision – input to higher-level layers
- Functional Implementation Layer – further interprets the touch decision input as a valid touch on a wheel or slider configuration
- User Application Layer – translates the data to defined User Interface functions.

Tool support

Renesas Electronics also provides a range of hardware and software tools designed to facilitate rapid device evaluation and help speed the time-to-market of R8C/3xT-based designs. For example, the Renesas Touch Workbench allows engineers to simplify the process of evaluating and tuning hardware and software for optimum touch performance, resulting in time and cost savings benefits. This powerful and simple-to-operate tool can be connected either through the HEW Target Server, via the E8a emulator, or through the serial interface.

The ever-increasing end-user appetite for touch-enabled mobile devices, such as e-readers, tablets, and smart phones, is driving demand for smaller, thinner profile and higher performance touch-key systems operating at lower power for extended battery life – all at lower costs. How do we achieve all of these aspects? While software implementation is an option, MCUs with integrated touch sensing capabilities are the key. MCUs with dedicated touch sensor units, like those provided by Renesas Electronics, offer engineers the required ability for scanning, measurement, noise counter measures, environmental variations, and data transfer while maintaining a low average power consumption – helping them to overcome the challenges of CPU utilization.

SORT BY PRICE SEARCH FEATURE

Digi-Key's Sort by Price feature makes it easier for engineers and purchasers around the world to order top-quality product from Digi-Key.

Sort by Price is based on tier 1 or unit pricing and applies to all parts, all of Digi-Key's 82 international websites, and all Digi-Key supported currencies.

Customers are able to sort by price in ascending or descending order in addition to applying filters for product in stock, lead-free product, and RoHS Compliant product. An Advanced Sort option is also available and allows customers to request parts in specific quantities.

Customers can view pricing of products from Digi-Key's vast line card of suppliers at a glance and choose the parts that best fit their particular needs.

WWW.DIGIKEY.COM

RX600 32-bit MCU Demonstration Kit

RENESAS RX

Quick and easy with a wide range of connectivity options

BUY YOURS TODAY!
YRDKRX62N-ND

- 100MHz RX62N MCU
- 10/100 Ethernet, CAN, USB (Host and OTG), UART/SPI/I²C
- µSD Card, 128MB Serial Flash
- Prototype area and Expansion headers (Wi-Fi & Bluetooth)
- SEGGER J-Link lite debugger
- RTOS, Projects & Demos already ported:
  - Ethernet (TCP/IP), Audio, FPU, USB, Accelerometer, File system and more
- Extensive Ecosystem:
  - Middleware, Stacks, IDEs, RTOS, Compilers, books and Documentation

RX THE CORE DIFFERENCE

www.digikey.com/renesas-mcu

© 2011 Renesas Electronics America Inc.

RENESAS

AVR32 UC3 Audio Decoder Over USB

contributed by Atmel

Atmel provides detailed source code for an MP3 decoder that runs on its AVR32 UC3 MCU. This article explains in detail how to configure and use it.

This article will help the reader to use the AVR32® UC3 Audio Decoder over USB software. This software includes a software MP3 decoder, a file system, and USB host mass storage class support.

For more information about the AVR32 architecture, please refer to the appropriate documents available at http://www.atmel.com/avr32.

License

The MP3 audio decoder (MAD) is distributed under the terms of the GPL. The JPEG decoder (IJG) license can be found in the file /SERVICES/PICTURES/JPG/IJG/license.txt. The memory manager (dlmalloc) license can be found in the file /SERVICES/MEMORY/MEMORY_MANAGER/DLMALLOC/license.txt.

Requirements

The software referenced in this article requires several components:

- A computer running Microsoft® Windows® 2000/XP/Vista or Linux®
- AVR32 Studio and the GNU toolchain (GCC) or IAR Embedded Workbench® for the AVR32 compiler
- A JTAGICE mkII or AVROne! debugger

Theory of operation

Overview

Today, embedded MP3 decoders are everywhere for consumers listening to audio content on mobile devices.

MP3

MPEG-1 Audio Layer III, more commonly referred to as MP3, is a digital audio encoding format using a form of lossy data compression. Several bit rates are specified in the MPEG-1 Audio Layer III standard: 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, and 320 Kbps. The available sampling frequencies are 32, 44.1, and 48 KHz, and a sample rate of 44.1 KHz is almost always used. 128 Kbps bitrate files are slowly being replaced with higher bitrates such as 192 Kbps, with some being encoded up to the MP3 maximum of 320 Kbps.

A tag, in a compressed audio file, is a section of the file which contains metadata such as the title, artist, album, track number, or other information about the file's contents.

The chosen MP3 decoder here is MAD (libmad), a high-quality MPEG audio decoder. It currently supports MPEG-1 and the MPEG-2 extension to lower sampling frequencies, as well as the so-called MPEG 2.5 format. All three audio layers (Layer I, Layer II, and Layer III) are fully implemented. MAD does not yet support MPEG-2 multichannel audio (although it should be backward compatible with such streams).

Block diagram

Figure 1 shows the UC3 interfacing to the USB stick, and outputting the audio stream from the key to the external DAC. The user can control the player using a keypad, or running a customizable Human Machine Interface (HMI).

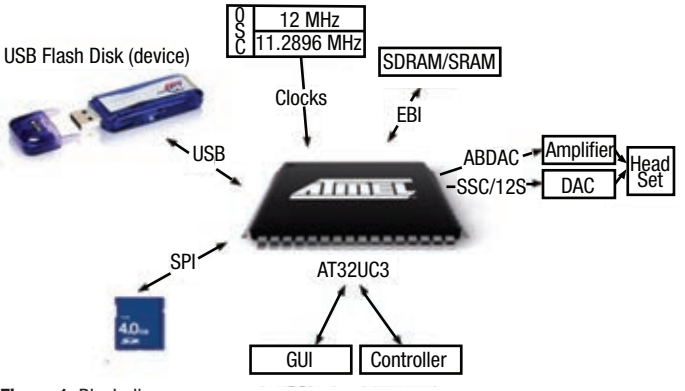


Figure 1: Block diagram.

Software architecture

Figure 2 shows the basic software architecture of the application.

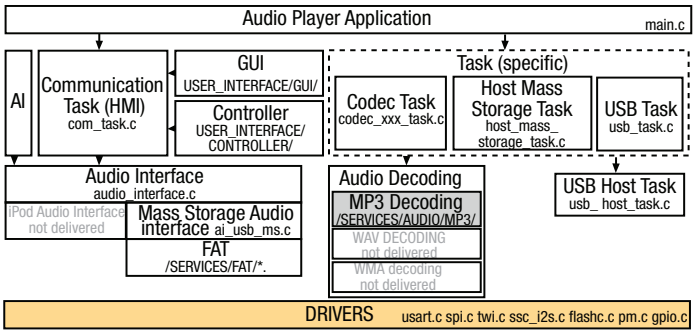


Figure 2: Software architecture.

The application does not require any operating system to run. The main() function is in charge of calling the software tasks (using a scheduler) that make audio decoding, HMI, and USB management possible. There are five tasks:

- The Communication Task contains the HMI of the application. This task holds the real intelligence of the user application and interfaces directly with the Audio Interface (AI). This is the task that the user should modify for their own application.
- Audio Interface Task: This task handles dynamic support for any newly plugged-in device types. For example, if a new device, using another class than the mass storage class, is plugged in, and if this class is supported by the application, then this task will tell the Audio Interface to link the Audio Interface API to a low level API that supports this class. All the mechanisms in place to dynamically link an API to this abstract API are located in the file: /SERVICES/AUDIO/AUDIO_PLAYER/audio_interface.[c|h]. In this application, only the USB mass storage class is implemented. Therefore, this function is not used.
- The last task is a macro used for feature tasks. In the MP3 audio player using USB, the modules involved will be the MP3 decoder, the USB stack, and the host mass storage task:
  - The MP3 Codec Task: This task is in charge of Audio Decoding management.
  - The USB Task: This task handles the USB stack and events.
  - The Host Mass Storage Task: This task will check for newly connected devices and initialize them using the USB mass storage class.

The main loop of the application is a simple free-running task scheduler:

```
while (TRUE)
{
      ai_task();
      com_task();
      task();
}
```

The Audio Interface can also support iPod audio decoding through the USB audio class. Please refer to the application note *AVR32730: AVR32 UC3 iPod Support Over USB* for more information.

Audio Interface API

The audio player is a generic audio player interface and is designed to support multiple audio formats besides MP3.

Overview

The Audio Interface manages disk navigation, audio navigation, and audio control. Thus, the user does not have to directly interface with the file system and audio control APIs. This greatly simplifies the software architecture.

The Audio Interface can be used in two ways:

- Using asynchronous functions, the results or effects of which may not be produced in a single iteration, usually leads to the use of state machines in the user firmware (since one must wait for the completion of a command before launching a new one), and has the advantage of reducing the risk of audio underrun. Asynchronous functions always have the "ai_async" prefix.
  Note that only one asynchronous function can be used at a time.
- Using synchronous functions, which are executed immediately, this drastically simplifies the user firmware architecture (no use of state machines since the synchronous AI functions are immediately executed), but may produce audio underruns since the execution time of these functions may be too long. Synchronous functions just have the "ai_" prefix.

All functions of the Audio Interface have a synchronous and asynchronous interface. For example, the command which returns the number of drives is:

- ai_async_nav_drive_nb() in the asynchronous interface
- ai_nav_drive_nb() in the synchronous interface

Using asynchronous functions shall be the preferred solution in order to avoid audio underruns.

Audio Interface (AI) architecture

The AI commands to interface the audio player are divided into three categories (See Figure 3):

- Disk navigation (Disk Nav) to browse into the tree architectures of the USB device
- Audio navigation (Audio Nav) to manage a list of playable songs
- Audio control (Audio CTRL) to control the audio stream (such as play, pause, and fast forward)

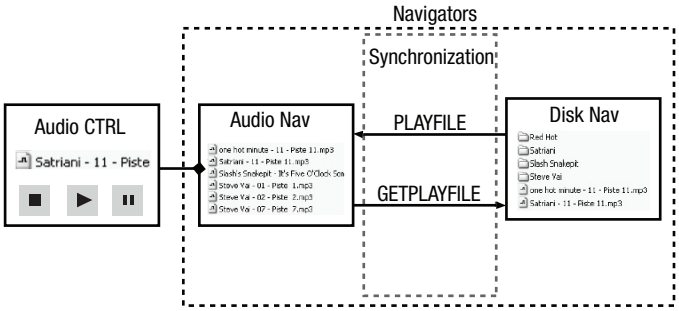


Figure 3: Audio player navigators overview.

Disk Navigator: "Browse into the tree architectures of the mass storage device"

The Disk Navigator is similar to a file explorer. It is used to navigate in the tree architectures of the connected USB device. It will hide all files whose extension differs from *.mp3, *.pls, *.smp, or *.m3u. The file extension filtering is automatically updated according to the modules selected (such as MP3 and playlists).

The Disk Navigator is totally independent from the Audio Navigator.

The commands associated with this module are used to navigate into the disks/directories, and to synchronize the selected file/folder with the Audio Nav module. This synchronization is made with two commands: ai_nav_getplayfile and ai_audio_nav_playfile.
See the "Disk Navigation" section later in this article for a complete list of commands.

Audio Navigator: "Manage a list of playable songs"

The Audio Navigator deals with a list of files. This list is defined once an ai_audio_nav_playfile command is executed. This command sets the list of files to be played according to the current selection in the Disk Navigator and to the Audio Navigator ai_audio_nav_expmode_set option.

When the ai_audio_nav_playfile command is executed:

- If the Disk Navigator is currently pointing to a playlist (*.m3u), then only the files of this playlist will be included in the Audio Navigator file list.
- If the Disk Navigator is currently pointing to a disk or a file, then the file list will depend on the audio explorer mode (set by the ai_audio_nav_expmode_set command) (See Table 1).
- If the Disk Navigator is currently pointing to a folder, the Audio Navigator will not enter into it. It will look for the first file that is in the current directory and build its file list according to the previous rules.
- If the Disk Navigator is not pointing to any files or folders (which is the case after a mount or a goto) when a directory or a playlist is selected, then it will select the first file of the file list.

Table 1: ai_audio_nav_expmode_set command behavior.

Explorer Mode	Behavior
AUDIO_EXPLORER_MODE_DISKS	Builds a file list off all playable songs of all disks, and starts playing from the selected file.
AUDIO_EXPLORER_MODE_DISK	Builds a file list off all playable songs of the current disk only, and starts playing from the selected file.
AUDIO_EXPLORER_MODE_DIRONLY	Builds a file list of all playable songs that are contained in the current directory, and starts playing from the selected file. Sub directories are ignored.
AUDIO_EXPLORER_MODE_DIRSUB	Builds a file list off all playable songs that are contained in the current directory and its sub-directories, and starts playing from the selected file.

Note that the playing order can be changed at compilation time by enabling a define. This is fully explained in the "Example 3 - Change the playing order" section later in this article.

Once the file list is set, two main commands are available to navigate into it:

- ai_audio_nav_next (select next file).
- ai_audio_nav_previous (select previous file).

Options can also be defined for the Audio Navigator (to change the repeat mode, or enable/disable the shuffle mode).

To synchronize the Disk Navigator with the Audio Navigator, i.e., to make the Disk Navigator select the file played by the Audio Navigator, the command ai_nav_getplayfile can be used. See the "Audio Navigation" section later in this article for a complete list of commands.

Audio Control: "control the audio stream (play/pause/fast forward/...)"

This module controls the audio stream of the selected file (selected by the Audio Navigator). Commands like play, pause, stop, fast forward, and fast rewind are available. See the "Audio Control" section later in this article for a complete list of commands (See Table 2).

Limitations

Speed

Speed navigation (such as file browsing) in directories may be affected if:

- A high-bitrate file is played at the same time.
- Directories have many files.
- The playlist includes many files.

Disk navigation

The exploration is based on a selector displacement. The file list is the list of the files in the current directory according to the extension filter (i.e., .mp3, .m3u, .pls, and .smp).

The file list:

- Is updated when you exit or enter a directory or a disk
- Starts with the directories then the files
- Is sorted in the creation order

Table 2: Disk Navigator commands.

Command Name	Input	Output		Description
		Return	Extra Result	
ai_get_product_id	–	Product ID	–	Returns the product identifier (USB PID) of the connected device (if available).
ai_get_vendor_id	–	Vendor ID	–	Returns the vendor identifier (USB VID) of the connected device (if available).
ai_get_serial_number	–	Length of the serial number in bytes	Serial number	Returns the serial number of the connected device (if available).
ai_nav_drive_nb	–	Number of drive	–	Returns the serial number of disk available.
ai_nav_drive_set	Drive Number	True or false	–	Selects the disk but does not mount it: (0 for drive 0, 1 for drive 1...). Returns false in case of error.
ai_nav_drive_get	–	Drive Number	–	Returns the selected disk number.
ai_nav_drive_mount	–	True or false	–	Mounts the selected disk. Returns false in case of error.
ai_nav_drive_total_space	–	Capacity of the drive	–	Returns the total space, in bytes and available on the device.
ai_nav_drive_free_space	–	Free space on the drive	–	Returns the free space, in bytes, available on the device.
ai_nav_dir_root	–	True or false	–	Initializes the file list on the root directory. Return false in case of error.
ai_nav_dir_cd	–	True or false	–	Enters in the current directory selected in file list. Return false in case of error.
ai_nav_dir_gotoparent	–	True or False	–	Exits current directory and goes to the parents directory. Then selects the folder from which it just exits, rather than selecting the first file of the parent directory. This simplifies navigation since the user firmware does not have to memorize the information. Returns false in case of error.
ai_nav_file_isdir	–	True or false	–	Returns true if the selected file is a directory, otherwise returns false.
ai_nav_file_goto	Position in the list	True or false	–	Goes to a position in file list (0 for position 0, 1 for position 1...). Returns false in case of error.
ai_nav_file_pos	–	File position	–	Returns the file position of the selected file in the current directory. Returns FS_NO_SEL if no file is selected.

Table 2: Disk Navigator commands. (continued)

Command Name	Input	Output		Description
		Return	Extra result	
ai_nav_file_nb	–	Number of audio files	–	Returns the number of audio files in the file list. Audio files are all the files which extensions matches *.mp3 or *.wma (it depends on the file format supported). There is a specific command for playlist files, see below.
ai_nav_dir_nb	–	Number of directories	–	Returns the number of directories in the file list.
ai_nav_playlist_nb	–	Number of playlists	–	Returns the number of playlists in the file list. The playlists are all the files matching with the extension *m3u, *.pls and *.smp.
ai_nav_dir_name	–	Length of the string in bytes	UNICODE name	Returns the name of the current directory.
ai_nav_file_name	–	Length of the string in bytes	UNICODE name	Returns the name of the selected file.
ai_nav_file_info_type	–	File type	–	Returns the type of the audio file selected.
ai_nav_file_info_version	_	Version number	–	Returns the version of the metadata storage method used for the selected audio file if available, otherwise, returns 0.
ai_nav_file_info_title	_	Length of the string in bytes	UNICODE title	Returns the title of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_artist	_	Length of the string in bytes	UNICODE artist	Returns the artist's name of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_album	_	Length of the string in bytes	UNICODE album	Returns the album's name of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_year	_	Year	–	Returns the year of the selected audio file if available, otherwise, returns 0.
ai_nav_file_info_track	_	Length of the string in bytes	UNICODE info	Returns the track information of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_genre	_	Length of the string in bytes	UNICODE genre	Returns genre of the selected audio file if available, otherwise, returns an empty string.
ai_nav_file_info_duration	_	Duration of the track	_	Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.
ai_nav_file_image	Max image size	Image size	-True or false -A pointer on the image data	Returns information and a pointer on the bitmap of the embedded cover art of the selected audio file. Returns false in case of error.
ai_nav_getoplayfile	_	True or false	–	Selects the file selected by the navigator. This command is the only link between these two navigators. Returns false in case of error.

Audio navigation

This navigator sets the list of files to be played. It can be seen as a playlist. Before accessing this navigator, an ai_audio_nav_playfile command must be issued.

Table 3: Audio Navigator commands.

Command Name	Input	Output		Description
		Return	Extra result	
ai_audio_nav_playfile	_	True or false	_	This command sets the audio file list according to the current mode of the audio navigator and plays (or sets to pause) the file selected in the disk navigator. In other words, it synchronizes the audio navigator with the disk navigator and plays (or sets to pause) the selected file from the disk navigator. Note that this command returns immediately and does not wait until the current track is completely played. This command does not change the current options (repeat/random/expmode). It is the opposite of the command ai_nav_getplayfile. Returns false in case of error.
ai_audio_context_save	_	Structure context	- True or false - The length of the structure in bytes	Gives complete audio context (player, state, play time, repeat, random, file played, explorer mode).
ai_audio_context_restore	Structure Context	True or false	_	Restores an audio context (eventually restart playing).
ai_audio_nav_next	_	True or false	_	Jump to next audio file available in the list. The next song file is chosen according to the current options (repeat/random/mode).
ai_audio_nav_previous	_	True or false	_	Jumps to previous audio file available in the list. The next song file is chosen according to the current options (repeat/random/mode).
ai_audio_nav_eof_occur	_	True or false	_	This routine must be called once a track ends. It will choose, according to the options (repeat/random/expmode), the next file to play.
ai_audio_nav_nb	–	Number of audio files present in the list	–	Returns the number of audio files present in the list.
ai_audio_nav_getpos	_	File position	_	Returns the file position of the selected file in the list.
ai_audio_nav_setpos	File position	True or false	_	Goes to a position in the list.
ai_audio_nav_repeat_get	–	Ai_repeat_mode	–	Returns the current repeat mode (no repeat, repeat single, repeat folder, repeat all).
ai_audio_nav_repeat_set	Ai_repeat_mode	–	–	Sets the current repeat mode (no repeat single, repeat folder, repeat all).
ai_audio_nav_shuffle_get	–	Ai_shuffle_mode	–	Returns the current shuffle mode (no shuffle, shuffle folder, shuffle all).

Table 3: Audio Navigator commands. (continued)

Command Name	Input	Output		Description
		Return	Extra result	
ai_audio_nav_ shuffle_set	Ai_ shuffle_ mode	–	–	Sets the current shuffle mode (no shuffle, shuffle folder, shuffle all).
ai_audio_nav_ expmode_get	–	Ai_ explorer_ mode	–	Returns the current explorer mode (all disks, one disk, directory only, directory + sub directories).
ai_audio_nav_ expmode_set	Ai_ explorer_ mode	–	–	Sets the current explorer mode (all disks, one disk, directory only, directory + sub directories). This mode cannot be changed while an audio file is being played.
ai_audio_nav_ getname	–	Length of the string in bytes	UNICODE name	Returns the name of selected file.
ai_audio_nav_ file_info_type	–	File type	–	Returns the type of the audio file selected.
ai_audio_nav_ file_info_version	–	Version number	–	Returns the version of the metadata storage method used for the selected audio file if available, otherwise, returns 0.
ai_audio_nav_ file_info_title	–	Length of the string in bytes	UNICODE title	Returns the title of the selected audio file if available, otherwise, returns and empty string.
ai_audio_nav_ file_artist	–	Length of the string in bytes	UNICODE artist	Returns the artist's name of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_ file_info_album	–	Length of string in bytes	UNICODE album	Returns the albums name of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_ file_info_year	–	Year	–	Returns the year of the selected audio file if available, otherwise, returns 0.
ai_audio_nav_ file_info_track	–	Length of the string in bytes	UNICODE info	Returns the track information of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_ file_info_genre	–	Length of the string in bytes	UNICODE genre	Returns the genre of the selected audio file if available, otherwise, returns an empty string.
ai_audio_nav_ file_info_duration	–	Duration of the track	–	Returns the total time of the selected audio file in milliseconds if available, otherwise, returns 0.
ai_audio_nav_ file_info_image	Max image size	Image size	-True or false - A pointer on the image data	Returns information and a pointer on a bitmap of the embedded cover art of the selected audio file. Returns false in case of error.

Modes used by the interface

Ai_repeat_mode defines the repeat mode:

- AUDIO_REPEAT_OFF: no repeat
- AUDIO_REPEAT_TRACK: repeat the current track
- AUDIO_REPEAT_FOLDER: repeat the current folder
- AUDIO_REPEAT_ALL: repeat the list of files

Ai_shuffle_mode defines the shuffle/random mode:

- AUDIO_SHUFFLE_OFF: no shuffle
- AUDIO_SHUFFLE_FOLDER: shuffle into the current folder
- AUDIO_SHUFFLE_ALL: shuffle into the list of files

Ai_explorer_mode defines the explorer mode (see the "Example 3 - Change the playing order" section later in this article for more information).

Audio Control

The audio controller is used to control the audio stream of the audio file selected by the Audio Navigator.

Table 4: Audio Control commands.

Command Name	Input	Output		Description
		Return	Extra result	
ai_audio_ctrl_stop	–	True or false	–	Stops the audio.
ai_audio_ctrl_ resume	–	True or false	–	Plays or resumes play after an ai_audio_ctrl_pause or an ai_audio_ctrl_stop command.
ai_audio_ctrl_ pause	–	True or false	–	Pauses the audio.
ai_audio_ctrl_time	–	Elapsed time	–	Returns the elapsed time of the audio track being played.
ai_audio_ctrl_ status	–	Status	–	Returns the status of the audio controller (stop, play, pause, a new audio file is being played, the current folder has changed).
ai_audio_ctrl_ffw	Skip time	True or false	–	Fast forwards the audio until the skip time has been reached. Then, it will continue to play the rest of the audio file. The skip time passed in parameter is in second.
ai_audio_ctrl_frw	Skip time	True or false	–	Fast rewinds the audio until the skip time has been reached. Then, it will set the audio player in play mode. The skip time passed in parameter is in second.
ai_audio_ctrl_ start_ffw	–	True or false	–	Sets the audio player into fast forward mode. **Function not implemented yet.**
ai_audio_ctrl_ start_frw	–	True or false	–	Sets the audio player into fast rewind mode. **Function not implemented yet.**
ai_audio_stop_ ffw_frw	–	True or false	–	Stops fast forwarding/rewinding and set the audio player into the previous mode (play or resume). **Function not yet implemented.**

Using asynchronous or synchronous API

Using synchronous functions is straightforward. Once finished, synchronous functions return, with or without a result.

Using asynchronous functions is more complicated; they may not produce the requested task in a single shot. Thus, these functions need some other functions to operate properly:

- void ai_async_cmd_task (void): This function is the entry point of all asynchronous commands. It must be called to execute the current state of the internal state machine of the current asynchronous function.
- Bool is_ai_async_cmd_finished (void): This function returns TRUE if the last command is finished.
- U8 ai_async_cmd_out_status (void): Returns the status of the last executed command (CMD_DONE, CMD_EXECUTING or CMD_ERROR).

- U32 ai_async_cmd_out_u32 (void): If the last executed command should return a 32-bit result or less, this function will return this value.
- U64 ai_async_cmd_out_u64 (void): If the last executed command should return a 64-bit result, this function will return this value.
- U16 ai_async_cmd_out_SizeArrayU8 (void): If the last executed command should return an extra result (e.g., a song name), this function returns the size in bytes of the extra result (no size limit).
- U8 ai_async_cmd_out_PtrArrayU8 (void): Returns a pointer to the extra result (assuming that the last executed command returns an extra result). This pointer can be freed by the application with the ai_async_cmd_out_free_ArrayU8() function.
- void ai_async_cmd_out_free_ArrayU8 (void): This function may be called to free the allocated buffer that holds the extra result. Note that the Audio Interface will automatically do this before executing a new command that needs extra results. This ensures that the application will not have memory leakage. Allowing the application calling this function to free the extra results sooner and improve allocated memory usage. Figure 4 shows the flow of the asynchronous function use.

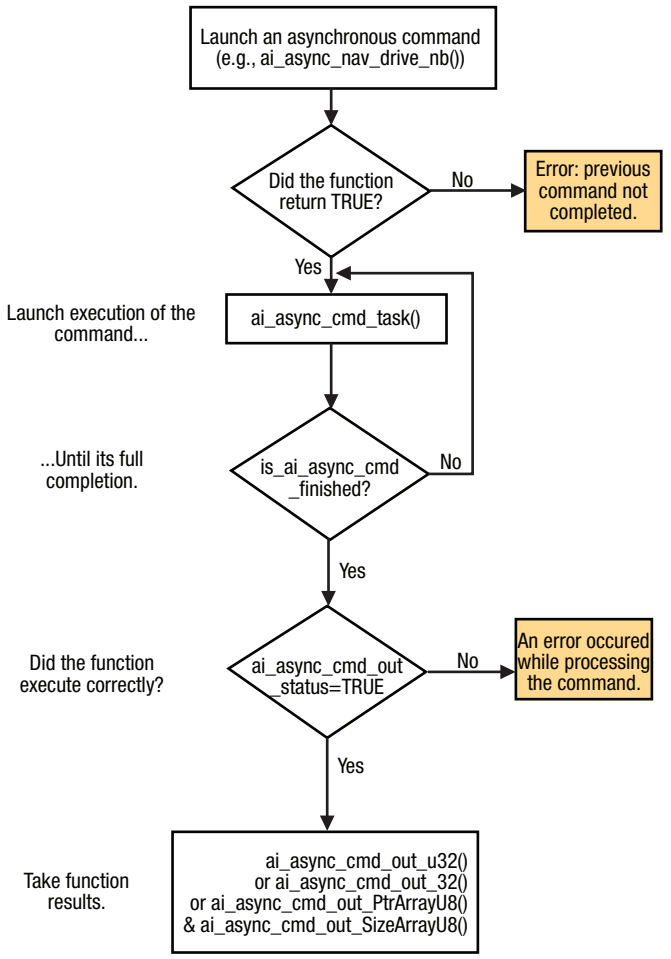


Figure 4: Asynchronous function flow.

Examples

The code for the next a three examples can be found in the following directories:

Let's take this disk as disk number 0 for the system.

Example 1 - Play "file1.mp3"

Table 5: Example: Play "file1.mp3."

Command Order	Command Name
0	ai_nav_drive_nb(): returns 1 disk.
1	ai_nav_drive_set(0): selects the disk 0.
2	ai_nav_drive_mount(): mounts the select disk 0.
3	ai_nav_file_goto(0): goes to file position 0
4	ai_nav_file_name(): returns the name dir1
5	ai_nav_file_isdir(): returns true, the current file is a directory.
6	ai_nav_file_goto(1): goes to file position 1
7	ai_nav_file_name(): returns the name dir2
8	ai_nav_file_goto(2): goes to file position 2
9	ai_nav_file_name():returns the name file1.mp3
10	ai_audio_nav_playfile(): selects the file selected by the file nagivator (file1.mp3)
11	ai_audio_ctrl_resume(): plays the selected file.
12	Wait for 10 seconds to listen to the beginning of the playback.
13	ai_nav_file_goto(3): goes to file position 3
14	ai_nav_file_name(): returns the name file2.wma

Example 2 - Play while browsing

Table 6: Example: Play while browsing.

Command Order	Command Name
0	ai_nav_drive_nb(): returns 1 disk.
1	ai_nav_drive_set(0): selects the disk 0.
2	ai_nav_drive_mount(): mounts the select disk 0.
3	ai_audio_nav_playfile(): selects a file in the audio navigator. By default it will seek inside the directories to play the first file which is "file5.wma" in our case.
4	ai_audio_ctrl_resume (): plays the selected file.
5	ai_nav_getplayfile(): synchronizes the disk navigator with the audio navigator. Now the disk navigator is pointing on the "file5. wma" file.
6	ai_nav_file_nb()+ai_nav_dir_nb()+ai_nav_playlist_nb(): gets the total number of entries (files+folder+playlist) in the current directory (dir3).
7	ai_nav_file_goto(0): goes to the file position 0
8	ai_nav_file_name(): returns the name "file5.wma". (Notice the difference with Table 5 - step #4)
9	ai_nav_file_goto(1): goes to file position 1
10	ai_nav_file_name(): returns the name "file6.mp3"
11	ai_audio_ctrl_stop(): stops the audio

Example 3 - Change the playing order

The playing order can be changed at compilation time by enabling the NAV_AUTO_FILE_IN_FIRST define (see the "Project Configuration" section later in this article).

Table 7: Playfile sequence.

Command Order	Command Name
0	ai_nav_drive_nb(): returns 1 disk.
1	ai_nav_drive_set(0): selects the disk 0.
2	ai_nav_drive_moutn(): mounts the select disk 0.
3	ai_audio_nav_playfile(): selects a file in the audio player.
4	ai_audio_ctrl_resume(): plays the selected file.

If the NAV_AUTO_FILE_IN_FIRST define is not set, the sequence will play audio files in the order shown in Table 8.

Table 8: Playfile sequence with NAV_AUTO_FILE_IN_FIRST undefined.

Order	File name	Parent directory path
0	file5.wma	/dir1/dir3/
1	file6.mp3	/dir1/dir3/
2	file4.mp3	/dir1/
3	file7.mp3	/dir2/
4	file1.mp3	/
5	file2.wma	/
6	file3.wma	/

If this define is set, the sequence will play audio files starting with files on the root, as shown in Table 9.

Table 9: Playfile sequence with NAV_AUTO_FILE_IN_FIRST defined.

Order	File name	Parent directory path
0	file1.mp3	/
1	file2.wma	/
2	file3.wma	/
3	file4.mp3	/dir1/
4	file5.wma	/dir1/dir3/
5	file5.mp3	/dir1/dir3/
6	file7.mp3	/dir2/

Source code architecture

Package

AUDIO-PLAYER-<board(s)>-<feature(s)>-<version>.zip contains projects for GCC (or AVR32 Studio) and IAR.

The default hardware configuration of the project is to run on Atmel® AVR32 UC3 Evaluation Kits, although any board can be used (refer to the "Board Definition Files" section later in this article).

Documentation

For full source code documentation, please refer to the auto-generated Doxygen source code documentation found in:

- /APPLICATIONS/AUDIO-PLAYER/readme.html

Projects/compiler

The IAR project is located in:

- /APPLICATIONS/AUDIO-PLAYER/<part>-<board>-<feature(s)>/IAR/

The GCC makefile is located in:

- /APPLICATIONS/AUDIO-PLAYER/<part>-<board>-<feature(s)>/GCC/

An AVR32 Studio project can be easily created by following the steps in the application note AVR32769: How to Compile the Standalone AVR32 Software Framework in AVR32 Studio V2.

Implementation details

This section describes the code implementation of the MP3 audio player running on the EVK1105. Other available packages are similar, so you will find useful information here that applies to every project configuration.

Main()

The main() function of the program is located in the file:

- /APPLICATIONS/AUDIO-PLAYER/main.c

This function will:

- Initialize audio output - refer to the "Audio Rendering Interface" section later in this article.
- Do the clock configuration.
- Call the USB Task - refer to the "USB" section later in this article.
- Call the USB Host Mass Storage Task. This task will check for newly connected devices and initialize them using the USB mass storage protocol.
- Call the Communication Task (HMI) - refer to the "HMI Communication Task Example" section later in this article.
- Call the Decoder Task to perform MP3 decoding - refer to the following section "MP3 Decoder."

The /APPLICATIONS/AUDIO-PLAYER/ contains the following files:

- ./main.c: Contains the main() function.
- ./com_task.c,h: HMI core. See the "HMI Communication Task Example" section later in this article for more details.
- ./USER_INTERFACE/CONTROLLER/joystick_controller.c: HMI using a joystick interface as a controller.
- ./USER_INTERFACE/GUI/et024006dhu_gui.c: HMI using an LCD screen as a display.
- ./codec_mp3_task.c: Handles the MP3 decoding task.
- ./CONF/*.h and ./<part>-<board>-<features>/conf_audio_player.h: Configuration file for audio, communication interface, memory and navigation explorer. Please refer to the "Project Configuration" section later in this article for more information on the configuration files.

MP3 decoder
The MP3 decoder source files are located in:

- /SERVICES/AUDIO/MP3/LIBMAD/: AVR32 port of LibMAD MP3 decoder.

A library of the decoder is provided in /UTILS/LIBS/LIBMAD/AT32UC/.

ID3 is supported up to version 2.4. The ID3 reader source is located in:

- /SERVICES/AUDIO/MP3/ID3/reader_id3.c,h.

Audio player API
The Audio Interface API is located in:

- /SERVICES/AUDIO/AUDIO_PLAYER/audio_interface.h

The Mass Storage Audio Interface can be found in:

- /SERVICES/AUDIO/AUDIO_PLAYER/AI_USB_MS/
  - ./ai_usb_ms.c,h: Mass Storage Audio interface.
  - ./ai_usb_ms_mp3_support.c,h: Adds support to the MP3 file format in the audio interface.
  - ./host_mass_storage_task.c,h: USB host mass storage task.

Refer to the "Audio Interface API" section earlier in this article for more details.

HMI communication task example
The included firmware implements an HMI example using a joystick and an SPI-driven LCD (source code is located under /APPLICATIONS/AUDIO-PLAYER-MASS-STORAGE/USER_INTERFACE/).

All the HMI is based on a pair of files, com_task.[c|h], which implement all mechanisms used to communicate between the internal APIs of the audio player and the user's HMI. An abstraction layer is used to easily attach all kinds of controller and graphical user interfaces to this communication task. This has been done to easily port the application to another board.

Controller
The controller is the module that makes the link between the driver and the abstraction layer, and is used to ensure compatibility with the communication task. It is based mainly on two API groups: "setup" and "events." All the APIs are defined in the file /APPLICATIONS/AUDIO- PLAYER/USER_INTERFACE/CONTROLLER/controller.h.

The function *controller_init* is used to initialize the controller. It may or may not be used, depending on the controller implementation.

All the other functions are Boolean functions, returning TRUE if an event has been raised or FALSE otherwise. Their names are explicit and follow the following naming convention:

Bool controller_<view>_<event>(void);

where *<view>* corresponds to the current view when the event should be taken into account. If no value is set for this field, the event is applied to all views.

The *<event>* is a short name describing the current event to which the function applies.

This example uses a joystick controller. The source code relative to this module is available in the file /APPLICATIONS/AUDIO-PLAYER/USER_INTERFACE/CONTROLLER/joystick_controller.c.

Graphical user interface
This module is used to display the high level audio player data to the user. It acts as a display and, when combined with the controller, it provides the user full control of the application.

This module is similar to the controller in terms of API groups.

The initialization function is called gui_init and includes all the parameters and frequencies needed to initialize the display.

This function is called at every occurrence of the main loop, at which point it resets the parameter flags which indicate parts of the view that need to be updated. Resetting these flags enables the GUI module to update the display asynchronously without having to update every component at once.

The example provides a graphical LCD display module that implements both a navigation and a playback view. All the code is based on the files /APPLICATIONS/AUDIO-PLAYER/USER_INTERFACE/GUI/[et024006dhu_gui.c|sdram_loader.c].

AT32UC3A drivers
The firmware uses the AVR32 UC3 driver library available in:

- /UTILS/LIBS/DRIVERS/AT32UC3A/.

The source code can be found in /DRIVERS.

USB
The USB low level driver is located in:

- /DRIVERS/USB/.

The USB mass storage service is located in:

- /SERVICES/USB/CLASS/MASS_STORAGE/.

FAT file system
The FAT12/16/32 files are located in the directory /SERVICES/FAT/.

Board definition files
The application is designed to run on Atmel Evaluation Kits. All projects are configured with the following define: BOARD=EVKxxxx. The EVKxxxx definition can be found in the /BOARDS/EVKxxxx directory.

Board customization

- For an IAR project, open the project options (Project -> Options), choose the "C/C++ Compiler," then "Preprocessor." Modify the "BOARD=EVKxxxx" definition by "BOARD=USER_BOARD."
- For GCC, just modify in the "config.mk" file (/APPLICATIONS/AUDIO-PLAYER/<part>-<board>-<feature(s)>/GCC/) the DEFS definition with "-D BOARD=USER_BOARD."
- For AVR32 Studio, open the project properties (Project -> Properties), go in the "C/C++ build," then "Settings," "tool settings" and "Symbols." Modify the "BOARD=EVKxxxx" definition by "BOARD=USER_BOARD."

The HMI can be easily changed. See the "HMI Communication Task Example" section earlier in this article for more details.

Audio rendering interface

The audio DAC mixer source code is located in /SERVICES/AUDIO/AUDIO_MIXER/audio_mixer.c,h.

I²S using the SSC module

The /COMPONENTS/AUDIO/CODEC/TLV320AIC23B/ directory contains the driver for the external DAC TLV320AIC23B.

The UC3 communicates with the TLV320AIC23B with the Two Wire Interface (TWI). The UC3 is the TWI master.

The AVR32 SSC module generates I²S frames using internal DMA (PDCA) to free CPU cycles for audio decoding.

Each time a new song is played, the SSC module is configured corresponding to the sample rate of the new song. The SSC clocks are composed of a bit clock and a frame sync:

- The bit clock is the clock used to transmit a bit from the audio stream. For a 44.1 KHz sample rate, the bit clock will be 44,100 x 2 (stereo) x 16 (bits per channel), i.e., 1.411 MHz.
- The frame sync is equal to the sample rate frequency, i.e., 44.1 KHz, taking the same example.

To get accurate 44.1 KHz audio frequency samples, an external 11.2896 MHz oscillator is used as input to an internal PLL, providing the CPU/HSB/PBA/PBB with 62.0928 MHz.

The TLV320AIC23B uses a master clock (MCLK) of 11.2896 MHz, outputted by the UC3 through a generic clock. Then, the generic clock output (PA07) is connected to the MCLK of the TLV320AIC23B.

The SSC clock divider in the CMR register is given by:

$$SSC.CMR.DRIV = Round\left(\frac{F_{PBA}}{2 \times (F_{SampleRateSetPoint} \times NumberChannel \times BitsPerSamples)}\right)$$

The real frequency sample rate outputted by the SSC is given by:

$$F_{AcutalSampleRate} = \left(\frac{F_{PBA}}{(2 \times SSC.CMR.DIV \times NumberChannel \times BitsPerSamples)}\right)$$

Note: NumberChannel = 2, BitsPerSamples = 16, F_{PBA} = 62.0928 MHz.

The music interval in semitones is:

$$MusicInterval(semitones) = \mathrm{LOG}\left(\frac{\left(\frac{F_{ActualSampleRate}}{F_{SampleRateSetPoint}}\right)}{\left(2^{\frac{1}{12}}\right)}\right)$$

Table 10: Sample rate with SSC module.

Sample Rate Set Point (Hz)	Actual Sample Rate (Hz)	Relative Error (%)	Music Interval (semitones)
8000	8018	0.23	0.04
11025	11025	0.00	0.00
16000	15095	-0.59	-0.10
22050	22050	0.00	0.00
32000	32340	1.06	0.18
44100	44100	0.00	0.00
48000	48510	1.06	0.18

For further information, please refer to the application note AVR32788: AVR®32 How to use the SSC in I²S mode.

ABDAC

The ABDAC driver is located in /DRIVERS/ABDAC.

The external amplifier driver (TPA6 13 0A) is located in /COMPONENTS/AUDIO/AMP/TPA6130A.

The AVR32 ABDAC module uses the internal DMA (PDCA) to free CPU cycles for audio decoding.

To get accurate audio frequency samples, the two external 12 MHz (OSC1) and 11.2896 MHz (OSC0) oscillators are used to source (directly or through a PLL) the ABDAC generic clock.

Table 11: Sample rate with ABDAC module.

Sample Rate Set Point (Hz)	ABDAC Generic clock input frequency	Actual Sample Rate (Hz)	Relative Error (%)	Music Interval semitones
8000	PLL0	8085	1.06	0.18
11025	OSC1	11025	0.00	0.00
16000	PLL1	15625	-2.34	-0.41
22050	OSC1	22050	0.00	0.00
32000	PLL1	31250	-2.34	-0.41
44100	OSC1	44100	0.00	0.00
48000	PLL1	46875	-2.34	-0.41

The PLL0 output frequency is 62.0928 MHz. The PLL1 output frequency is 48 MHz (used for USB).

When used, the ABDAC generic clock divider is given by:

$$ABDAC\ generic\ clock\ divider = Round\left(\frac{F_{GCLKInput}}{(2 \times 256 \times (F_{SampleRateSetPoint}}\right) - 1$$

Refer to /DRIVERS/ABDAC/abdac.c for the configuration of the ABDAC generic clock.

SDRAM loader

Graphical data, such as images, consumes a lot of space in RAM and flash which could be used by the application instead. Because of that, the images used by the audio player application's GUI are stored in DataFlash and are loaded to SDRAM upon startup. The reason for having the data in the SDRAM, rather than in flash, is the speed improvement which leads to faster updates of the display with new content.

The SDRAM loader module uses a memory manager to manage the SDRAM memory space. This memory manager can also be used in the application to allocate memory from SDRAM, instead of the internal SRAM.

The following sections give a short introduction to the parts involved in the SDRAM loader module.

The SDRAM loader source code is located in /APPLICATIONS/AUDIO-PLAYER/USER_INTERFACE_GUI/sdram_loader.c,h.

DataFlash

The DataFlash is formatted with a FAT16 file system, and it currently contains graphical data used by the audio player application in the GUI implementation. The graphical data is stored as BMP files and the format is RGB565. This can be used directly on the display when swapped from little endian to big endian. Other BMP formats are currently not implemented but this could be extended, if needed.

The FAT file system makes it easy for developers to upgrade the content with new files or even use it for other applications such as a web server. An upgrade of the content can be done by using a mass storage example application.

The BMP pictures used in the application are stored in the /PICTURES directory.

To customize the GUI, the bitmap files can be updated and must be saved to the Windows bitmap image format using 16-bit R5 G6 B5 encoding. This can be done using GIMP, the GNU Image Manipulation Program (see http://www.gimp.org/).

Loading process
The SDRAM loader consists of two files, sdram_loader.c and sdram_loader.h, in the directory APPLICATIONS/AUDIO-PLAYER/USER_INTERFACE/GUI. In these files, the images that should be loaded to SDRAM and how they should be converted are specified. A configuration that loads three images to SDRAM could look like this:

```
typedef struct {
const wchar_t *name;
void *start_addr;
} ram_file_t;
...
#define STARTUP_IMAGE 0
#define DISK_NAV_IMAGE 1
#define AUDIO_PLAYER_IMAGE 2
#define NUMBER_OF_IMAGES 13

ram_file_t ram_files[NUMBER_OF_IMAGES] = {
    { .name = L"/AVR32_start_320x240_RGB565.bmp"},
    { .name = L"/disk_nav_320x240_RGB565.bmp"},
    { .name = L"/audio_player.bmp"}
};
```

The ram_files array is used throughout the GUI to get access to the image data. In order to load other data than RGB565 BMP data to SDRAM, the module needs to be modified.

The SDRAM loader module is called once during the initialization of the graphical user interface. When called, it initializes the SDRAM interface and reads the raw image data from specified BMP files into SDRAM. To get the raw image data, the BMP header must be read to get the image size and the offset of the data in the file. The copy process also does a conversion from little endian to the big endian data ordering, thus the final image data can be dumped directly into the display buffer.

A single call to the SDRAM module is enough to do the initialization and load process:

```
void load_sdram_data(int hsb_hz);
```

The sole parameter is the HSB frequency in hertz, which is needed to initialize the SDRAM timings. The load_sdram_data function is called starting from main in the following order:

```
main() -> com_task() -> gui_init() ->
sdram_load_data()
```

SDRAM memory management
The images could be placed at specified locations in SDRAM, making memory management unneeded, but it is often better to do memory management to remove the task of keeping track of the data in memory from the developer.

The audio player uses a separate memory manager to manage SDRAM memory (this memory manager can also replace the default memory manager in the Newlib library if needed).

The source files of the memory manager are located in /SERVICES/MEMORY/MEMORY_MANAGER/DLMALLOC/ and the additional configuration of it is /APPLICATIONS/AUDIO-PLAYER/CONF/conf_dlmalloc.h.

The memory manager is initialized in the SDRAM loader module and it is configured to use the whole SDRAM memory. After the initialization, memory can be allocated from the SDRAM with the mspace_malloc call. Memory from the internal SRAM can be allocated with the default malloc call.

JPEG decoder
The JPEG decoder is used for MP3 cover art file support.

The source code of the IJG JPEG decoder is located under /SERVICES/PICTURES/JPG/IJG/. Documentation for the library can be found in /SERVICES/PICTURES/JPG/IJG/libjpeg.doc and an overview is located in the README file.

The IJG license text can be found in /SERVICES/PICTURES/JPG/IJG/license.txt.

The JPEG decoder application source files are located under /APPLICATIONS/AUDIO-PLAYER/JPG/. This is a layer that handles the data input and output of the JPEG library. It also handles the error handling to jump out of the library in case of a critical error. Since the JPEG decoding is done in SDRAM, the memory management backend of the library, which normally would be included in the library itself, is located here. The memory management backend is modified to use SDRAM with the DLMALLOC memory manager (in /SERVICES/MEMORY/MEMORY_MANAGER/DLMALLOC/) instead of the standard malloc implementation.

Project configuration
There are two different configuration files. The one related to the audio player itself, located under /APPLICATIONS/AUDIO-PLAYER/<part>-<board>-<feature(s)>/conf_audio_player.h, is mostly a high level configuration file, but can be customized to support certain configurations or enable/disable audio player features. The rest of the configuration files are located under the /APPLICATIONS/AUDIO-PLAYER/CONF/ directory and should be modified to change, separately, the audio player's module configurations.

Configuration files are not linked to IAR, GCC, or AVR32 Studio projects. The user can alter any of them, then rebuild the entire project in order to reflect the new configuration.

High level configuration file: /APPLICATIONS/AUDIO-PLAYER/<part>-<board>- <feature(s)>conf_audio_player.h

- DEFAULT_DACS specifies the default audio DAC used for the audio output. Three values are possible: AUDIO_MIXER_DAC_AIC23B for the I²S interface, AUDIO_MIXER_DAC_ABDAC for the internal DAC (ABDAC module), and AUDIO_MIXER_DAC_PWM_DAC to use PWM channels (an external low-pass filter is required).

- USE_AUDIO_PLAYER_BUFFERIZATION, set to “ENABLED” to support the navigation while playing feature, will use extra memory to buffer decoded samples in order to prevent audio blips while the user navigates in the disk architecture, for example. The memory address and size can be configured in the file “conf_buff_player.h.” Note that using the audio sample bufferization will not ensure that every audio blip will be covered. It will always depend on the speed of the USB device connected, its file system, and the operations requested. The default buffer size is set to 128 Kbytes.
- SUPPORT_MP3, SUPPORT_PLAYLISTS, SUPPORT_EMBEDDED_COVER_ARTS is a set of defines which can be enabled or disabled to reduce audio player features. Note that the use of the embedded cover art requires RAM in order to decode embedded JPEG pictures. Therefore, an external memory is needed to handle this feature.

Low-level configuration files: /APPLICATIONS/AUDIO-PLAYER/CONF/*.h

conf_access.h – This file contains the possible external configuration of the memory access control. It configures the abstract layer between the memory and the application, and specifies the commands used in order to access the memory. For example, this file will define the functions to be called for SD/MMC memory access.

conf_audio_interface.h – A set of configuration flags to enable/disable internal features of the audio player.

conf_audio_mixer.h – Configures all parameters relative to the audio DACs. This file is made to support multiple configurations, and can be easily upgraded to handle new DACs.

conf_buff_player.h – Defines the starting address and the size of the memory used to bufferize audio samples (if the feature is enabled).

conf_dmalloc.h – Configuration of malloc/free functions.

conf_explorer.h – It defines the configuration used by the FAT file system. The configuration is also applied to the playlist handler and the file navigation. The main parameters are:

- NAV_AUTO_FILE_IN_FIRST: Must be defined in order to play first the files at the root of a directory instead of the ones inside the subdirectories.
- FS_NAV_AUTOMATIC_NBFILE: This flag can be set to DISABLE in order to speed up the response of the ai_audio_nav_playfile command. On the other hand, the three commands ai_audio_nav_getpos, ai_audio_nav_getpos, and ai_audio_nav_nb will not be available anymore. It will also affect the use of the explorer modes, if different from “all disks” and “one disk.”

conf_jepg_decoder – Configuration of the JPEG decoder.

conf_pwm_dac.h – Configuration of the PWM DAC (which PWM channel is used, which pins are concerned).

conf_tlv320aic23b.h – Configuration of the external I2S DAC (which pins and which configuration interface are used).

conf_usb.h – Configuration file used for the USB.

conf_version.h – Internal version of the firmware.

Compiling the application

The following steps show you how to build the embedded firmware according to your environment.

If you are using AVR32 Studio:

- Please refer to the application note *AVR32769:* How to Compile the standalone *AVR32* Software Framework in *AVR32* Studio V2.

If you are using the standalone GCC with the AVR32 GNU Toolchain:

- Open a shell, go to the /APPLICATIONS/AUDIO-PLAYER/<partf>-<board>-<feature(s)>/GCC/ directory and type:
  make rebuild program run.

If you are using IAR Embedded Workbench for Atmel AVR32:

- Open IAR and load the associated IAR project of this application (located in the directory /APPLICATIONS/AUDIO-PLAYER/<part>-<board>-<feature(s)>/IAR/).
- Press the “Debug” button at the top right of the IAR interface.

The project should compile. The generated binary file is then downloaded to the microcontroller to switch to the debug mode.

- Click on the “Go” button in the “Debug” menu or press F5.

FAQ

Q: What is the maximum number of playlist links supported?
A: The file system supports up to 65,535 links inside a playlist.

Q: What are the supported text formats?
A: The file system supports the ASCII, UTF8 and UNICODE (UTF16LE & UTF16BE) text formats.

Q: How are the directories and files sorted inside the disk?
A: The logical structure is the same as an explorer view. Directory and files are sorted using their creation order.

Q: Which file systems are supported?
A: FAT12/16/32.

Q: What is the maximum number of files supported in a directory?
A: There is no limitation in the firmware for the supported number of files and directories. The only limitation is due to the FAT file system:

- for FAT12/16 root directory only: up to 256 files (short names).
- for FAT12/16/32 up to 65,535 files (short names) per directory.

Q: What is the minimum RAM requirement to run this application?
A: The default application is using external SDRAM to support all features. This application can run with 64 kbytes of RAM (internal size on AT32UC3A0512) by disabling the audio bufferization and the cover art support. Audio blips will be present if the user tries to navigate in the disk while a track is played. The SDRAM loader also uses external SDRAM memory for graphical data.

Q: What kind of license applies to the software?
A: Atmel provides the audio decoder software library free of charge. But some digital audio formats, including MP3 and WMA, contain technology that is patented, and a license for these patents must be obtained before the library can be used.

Developing Next-Generation Human Interfaces Using Capacitive and Infrared Proximity Sensing

by Steve Gerber, *Silicon Laboratories, Inc.*

"High tech/high touch" took on new meaning with the advent of touchscreens. Adding IR proximity sensing takes it to the next level.

Next-generation human interfaces, based on capacitive and infrared proximity sensing technology, can provide dramatic improvements in the end-user experience while increasing system reliability and reducing total cost. In addition to making electronic products easier to use and more visually appealing, these interfaces mask the increasing underlying complexity of electronics, enabling manufacturers to bring products with advanced capabilities more quickly to the mass market.

Advanced sensor-based interfaces are more reliable than traditional mechanical interfaces since they do not contain the moving parts associated with buttons and dials, which are prone to failure over time. Sensor-based control panels and displays are also more flexible, allowing a single set of controls to be reconfigured based on application context so that users are presented with only those choices that are currently active. When combined with gesture recognition and "touchless" technology, developers can add intelligence to device interfaces that anticipates user needs and enables innovative usage models to make products friendlier and more intuitive to use. Flexible firmware can be adjusted quickly and easily as market needs change without having to completely re-architect systems or redesign device packaging.

Next-generation human interfaces

Next-generation products require next-generation human interfaces to differentiate themselves in the marketplace. By making electronic devices more aware of their environment, new features can be implemented which enhance ease of use, improve power efficiency, and reduce system cost. In addition, high sensitivity, low noise, and resistance to moisture ensure reliability, even in the most challenging environments.

Two of the primary technologies driving next-generation interface development are capacitive and proximity sensing. Capacitive sensing detects the presence of a human finger through changes in the capacitive value of the sensing element. It enables advanced controls such as sliders and wheels and is well-suited for close-range interfaces where users are used to physical feedback, such as when they press a button. Proximity sensing uses infrared sensors to calculate the distance of objects up to one meter away using infrared reflectivity techniques. Proximity sensors also can place an object in space, enabling "touchless" gesture tracking.

Using these two technologies together enables finer tuning of user interfaces. Many end users are already familiar with capacitive sensing technology from its use in a wide array of consumer products, most prominently the iPod and iPhone. Until recently, proximity sensing has typically been used for simple tasks such as cheek detection in handsets. However, it can be used for so much more:

- **User detection:** Proximity sensing can determine, for example, whether an end user is currently sitting at a PC and can turn off the display when the user walks away from the desk. Given the substantial power required for LCD backlighting, even simple user detection can result in significant power savings across an organization. User detection also can be implemented in devices such as USB chargers or thumb drives so that devices can prepare themselves for sudden removal.
- **Fingerprint-free displays:** Many portable devices require users to touch buttons all over the screen, leaving oily marks that are difficult to see through and clean. A touchless interface for a portable multimedia player, for example, would eliminate the need for users to touch the very screen on which they want to watch a video. Similar applications include enabling users to flip through the pages of an electronic cookbook with touchless ease, or allowing doctors to directly interact with touch screen-based systems during surgery without fouling the screen with fingerprints.
- **Automatic backlighting control:** Part of the proximity sensing signal path is the use of an ambient light sensor (ALS) to reduce noise from external light sources. This same sensor also can be used to monitor background lighting conditions and automatically adjust the display backlight appropriately to reduce power consumption.
- **Invisible intrusion detection:** Reflecting infrared light off the interior door surface of a system allows developers to deploy an "invisible" intrusion mechanism which avoids the unreliability and expense of mechanical switches used for the same purpose.
- **Health and safety benefits:** Kiosks, check-out stands, and other public computers present health risks in terms of spreading disease via keyboards or touch screens. In parts of China, for example, laws require that every elevator panel be wiped down once an hour to prevent the spread of SARS. Touchless panels avoid and mitigate these public health issues.

Offloading interface control

One emerging trend in embedded design is to offload user interface management from the primary application processor to a dedicated, 8-bit MCU. Human touch is a relatively slow event to an application processor, and powering the entire system to check if a user has moved his or her finger consumes significantly more power than is required by an 8-bit MCU to accomplish the same task.

Capacitive touch-sense MCUs, such as the F99x family from Silicon Labs, are ideally suited for managing next-generation user interfaces. By providing up to 25 MHz of performance combined with peripherals optimized for the task, F99x MCUs provide the processing and input capabilities necessary to implement intelligent and accurate sensing. When implemented in conjunction with Silicon Labs' Si114x family of proximity sensors, developers can implement highly efficient user interfaces within a single development environment.

Capacitive sensing performance for the F99x MCUs is further enhanced through a hardware-based, capacitive-to-digital converter (CDC). Developed by Silicon Labs, the CDC includes two current inputs which are digital-to-analog converters (DACs). The first is a variable DAC that measures the current to the external sensor capacitor, and the second is a constant current source for an internal reference capacitor (see Figure 1). Capacitance is measured using successive approximation registers (SAR) – an efficient process immune to DC offset which requires no external components.

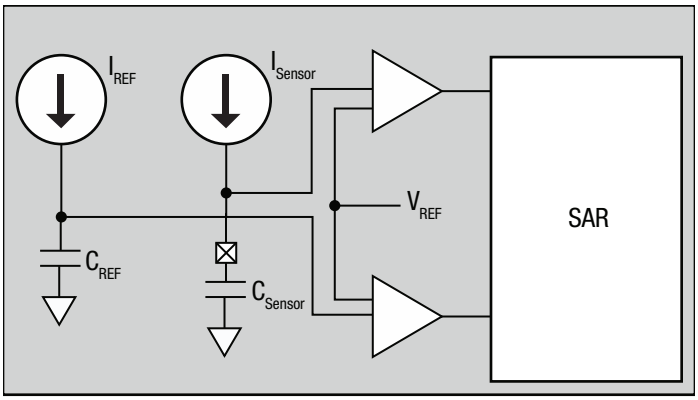


Figure 1: The hardware-based CDC enables high performance, 16-bit resolution, high reliability and DC offset immunity – without requiring external components.

The F99x MCU's 16-bit CDC offers high reliability and accuracy. By performing a two-stage discharge of the external capacitor, the CDC can remove ambient noise energy captured during the discharge process. In comparison, other approaches require additional external components (e.g., series resistors) and more than one I/O per channel (thereby increasing MCU size and increasing routing difficulty).

The CDC's dynamic range is improved through the use of an adjustable gain. The dynamic range is also enhanced by the ability to reduce the source current to change the charge timing and more directly reflect the voltage at the capacitive sensor when the source current and series impedance are both high (such as when using a touch panel or ESD-protected capacitive pads). Higher sensitivity gives developers greater signal margin, allowing them to use thicker plastics, make electrodes smaller, and operate reliably in noisy environments. The CDC also employs pin monitoring to automatically adjust conversion timing if necessary to eliminate possible interference from high-current switching on nearby pins. In summary, the CDC enables a superior signal-to-noise ratio (SNR) of between 50 to 100 for a typical capacitive sensing implementation.

Unmatched system responsiveness

Proximity sensing employs an infrared sensor and one or more infrared light-emitting diodes (LEDs). The basic operating principle is to illuminate an object and measure the intensity of the reflected light. The number of LEDs required depends upon the application and what spatial information is needed. A paper towel dispenser sensor, for example, needs only a single LED to detect that a person is standing in front of the dispenser. To detect a left/right or up/down gesture, two LEDs are required. To support full spatial navigation, three LEDs are needed. In each case, only a single physical sensor is required. However, each additional sensor increases the processing required to identify the intensity of signals received from each LED and to triangulate the detected object's position.

Processing is also required to filter noise (i.e., background light) from the received signal. The more powerful the processor or embedded controller, the more samples it can take and the better filtering it can achive. Increasing the sample rate increases the resolution of the system, while better filtering increases accuracy. Rapid sampling and precise filtering are both required for a robust interface, and developers must balance each approach to optimize their application.

Typically, the extended acquisition times associated with low-sensitivity photodiodes allow flicker from sources such as fluorescent lights to reduce accuracy. Silicon Labs' high-sensitivity photodiode technology – proven in the industry for more than ten years – exhibits excellent immunity to EMI and flicker while reliably detecting objects up to 50 cm away without the use of external lenses or filters. Built upon this robust photodiode technology, the Si114x sensor family supports ambient light sensing capabilities.

The primary power drain in a proximity sensing subsystem is the infrared LED. Silicon Labs' QuickSense development environment assists developers in defining configuration parameters to optimize accuracy, detection range, and power consumption. For example, advanced control capabilities allow developers to dynamically adjust LED current to tune it for a particular application and detection range. Developers also can control the refresh strobe rate to further minimize LED power consumption. For ultra-low-power operation, developers can use innovative single-pulse proximity sensing to minimize LED on-time and achieve up to a 4000 times improvement in power consumption, as shown in Figure 2.

Reducing system power consumption

With the current emphasis on green, energy-saving electronics, all devices, not just portable devices, are beginning to be designed with power conservation in mind. Part of an effective low-power strategy is to minimize CPU active time while maximizing sleep time for as many components as possible within the system. Silicon Labs has implemented several mechanisms for reducing overall system power consumption in its capacitive touch-sense MCUs:

- **Background scanning:** Since the CDC is implemented in hardware, capacitance measurements for channel scanning can be completed autonomously while the CPU operates in its power-saving suspend mode.

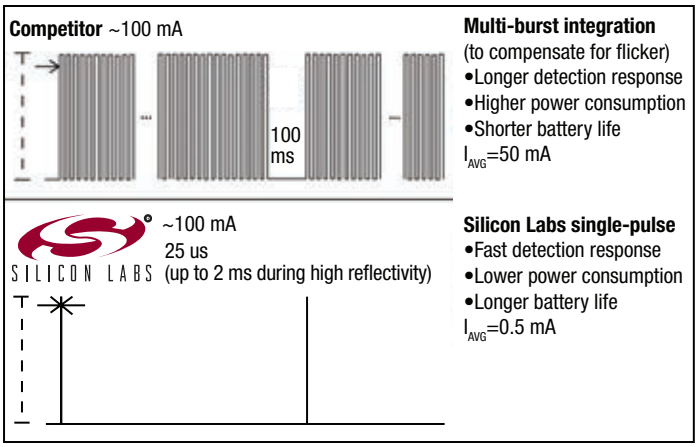


Figure 2: QuickSense MCUs offer innovative single-pulse proximity sensing to minimize LED on-time and reduce power consumption by up to 4000 times.

- **Autonomous auto scanning:** Rather than scan and convert all capacitive sensing channels, only active channels are scanned and converted.
- **Channel bonding:** Scanning several channels together using a single input consumes less power than handling multiple conversions to check channels individually. For example, the system can scan an entire slider using a single input and wake the CPU if any active channel is touched. Once awake, the CPU can then scan each channel separately to determine which channels were touched and begin interpretation of pending gestures.
- **Integrated LDO regulator:** The F99x MCU's integrated low drop out (LDO) voltage regulator provides linear response while maintaining a constant, ultra-low active current at all voltages. In addition, the F99x has special circuitry to retain RAM when the LDO regulator is disabled in sleep mode.
- **Flexible operating voltage:** For many MCUs, the CPU must be operated at a lower frequency as the operating voltage is reduced, thus increasing operating time and power consumption. MCUs limited to 2.2 V operation also waste 20 percent battery life when using AA/AAA batteries. With full operating capabilities at 25 MHz down to 1.8 V, the F99x maximizes battery powered efficiency across applications.

Most MCUs are designed to optimize either active or sleep power efficiency. The F99x architecture was designed from the ground up to offer the industry's lowest power in both active mode and sleep mode (see Table 1). An internal power management unit (PMU) limits leakage, resulting in active and sleep currents less than half that of the F99x's closest competitors.

Table 1: F99x active and sleep mode power consumption.

Mode	Current Consumption
Active	150 µA/MHz*
Sleep with brownout detection disabled	10 nA
Sleep with brownout detection enabled	50 nA
Sleep with internal RTC operational	300 nA

**When operating at 0.9 to 1.8 V, C8051F99x MCUs achieve even greater power efficiency through the use of an internal boost converter.*

Fast wake-on-touch

An important technique for reducing power consumption is to turn off a device's display and control interfaces when they are not in use and then place the entire system into sleep mode. A key element of interface design is how responsive the system is to a user when it is in transition between sleep and active modes, (i.e. how fast it can wake up). With a capacitive sensing-based system, when the system is asleep there is no backlighting to indicate to the user which function each capacitive button or slider currently represents. Thus, the first button press is limited to simply waking up the system.

When proximity sensing is available, a system can detect a user from up to one meter away. This capability allows the proximity sensor to wake the system as the user approaches or reaches for the device so the system can wake and the display can be fully functional by the time the user is ready to press a button. In practical application, this changes how users can interact with devices by making systems more intelligent and friendly. For example, devices such as car stereos or set-top boxes can have black panels that "disappear" from sight when not in use but turn on with full awareness when a user's hand is near.

Wake-up time is measured as the time interval between recognizing the need to wake and when the first instruction is executed. Wake-up time depends upon many factors, including regulator stabilization and analog settling time. For events such as reading a capacitive or proximity sensor, among the first actions the CPU must take is an analog measurement. If the analog peripheral is not ready to be read, this increases the effective wake-up time. Wake-up time not only defines system responsiveness, it also affects power efficiency. During wake-up, the MCU performs no work but still consumes power. Shorter wake-up time, therefore, reduces the power wasted as the CPU wakes up.

To complicate evaluating wake-up time, vendors measure wake-up time using different criteria. Some MCUs wake to trigger an interrupt service routine (ISR) and must wait before taking an analog measurement. Wake-up time, in these cases, is measured from the wake event to either an MCLK valid on the appropriate pin or when the interrupt vector is fetched. To obtain a wake-up time, equivalent to measurements made until the execution of the first code instruction, developers must add several µs/CPU cycles to the measurement.

The wake-up time of the F99x MCU has been speed optimized, resulting in a wake-up time from sleep of 2 µs. In addition, its analog settling time is only 1.7 µs, 15 times faster than competitive MCUs. Thus, the effective wake-up time from event to first analog measurement is less than 4 µs, up to 7 times faster than the closest competitor.

In addition to fast responsiveness, F99x MCUs have the industry's lowest power capacitive touch sensing available on the market today. They provide outstanding performance – 150 µA/MHz – over the entire 1.8 to 3.6 V operating range, as well as the industry's lowest power wake-on-touch at less than 1 µA. Fourteen CDC channels provide ultra-fast 40 µs acquisition time, 16-bit accuracy, built-in averaging to increase reliability, and interference immunity to low frequency noise and DC offsets. The F99x MCU's CDC is the fastest and most sensitive capacitive-to-digital converter available today, and other devices with equivalent sensitivity take more than 1000 times longer to sample. The highly programmable F99x MCUs enable

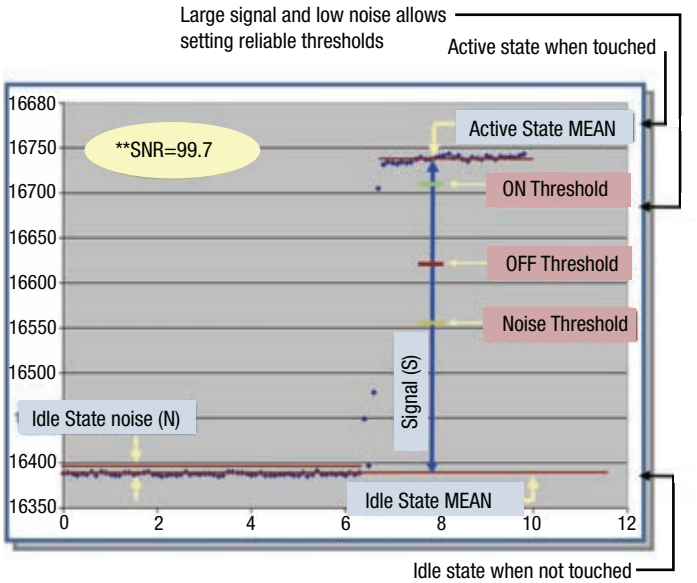


Figure 3: Developers can dynamically adjust active and inactive thresholds to accommodate changing environmental factors for superior sensing reliability.

developers to dynamically adjust active and inactive thresholds to accommodate changing environmental factors for superior sensing reliability (see Figure 3).

Silicon Labs' QuickSense portfolio includes a wide variety of sensing devices. In addition to the F99x MCUs, Silicon Labs' F8xx and F7xx MCU families provide advanced capacitive sensing capabilities, optimal performance, efficient power consumption, and low cost for a wide range of applications. For proximity sensing, developers can use Silicon Labs' Si114x infrared proximity and ambient light sensors. These proximity sensing devices offer power-saving, single-pulsing technology and touchless gesture support. Silicon Labs' infrared-based proximity sensors are the fastest sensing devices on the market, offering the longest sensing range without compromising power or efficiency.

Advanced development environment

As embedded applications continue to become more complex, designing a robust implementation requires not only proven hardware but also production-ready software and world-class development tools. To aid developers, Silicon Labs offers the QuickSense Studio, a comprehensive suite of hardware, software, and development tools to enable developers to quickly and easily introduce capacitive and proximity sensing to any application.

From an application perspective, both capacitive and proximity sensors can be thought of as simple inputs to the system. By abstracting their implementation through an API, developers can access user interactions, regardless of their source, as touch or gesture events which can be easily mapped to specific functionality, thus substantially simplifying application and interface development. The easy-to-use, GUI-based QuickSense Configuration Wizard accelerates development by generating all application configuration code and firmware drivers required, eliminating the need for developers to understand or write any lower level code for MCU peripherals used to monitor the sensors. Industry-proven firmware manages the different capacitive interface options – including touch button, slider, and wheel – and capacitive proximity sensors. Developers have full control over important sensor characteristics such as sensitivity, operation thresholds, responsiveness, and code size.

QuickSense Studio also automatically calibrates sensors and provides full debugging and performance analysis capabilities to ensure that designs are responsive and robust. Even if several switches are the same size and shape, for example, their location on the PCB will affect their active and inactive state capacitance given their proximity to other conductive elements, ground planes, and the presence of electrical interference. Each switch will need to be calibrated during development or production with the results programmed to flash memory. In addition, if effects from environmental factors such as temperature, humidity, supply voltage, and contaminants are large enough, incorrect measurements can lead to false sensor events. QuickSense Studio enables systems to account for the dynamic nature of these factors by periodically reconfiguring themselves.

QuickSense Studio is the only development tool on the market that supports both capacitive and proximity sensing, enabling developers to design a complete user interface using a single development environment. In addition to the Configuration Wizard, QuickSense Studio accelerates the design of the following:

- Infrared proximity sensing
- Ambient light sensing
- Capacitive buttons and sliders
- Capacitive proximity sensing
- Complex algorithms
- Gesture recognition
- MCU control and communications
- Capacitive touch screens

Summary

Effective human interfaces involve a combination of pleasing aesthetics and revolutionary ways to interact with electronic devices. Manufacturers seeking to differentiate their products with as little impact as possible on system cost and power consumption can turn to next-generation interfaces based on capacitive and proximity sensing to provide an easier, more intuitive user experience. By combining the capacitive and proximity sensing capabilities of devices such as Silicon Labs' C8051F99x capacitive touch-sense MCUs and Si114x proximity sensors, developers can introduce next-generation gesture and touchless interfaces to any system quickly and easily using industry-proven hardware and firmware.

For more information about capacitive touch and proximity sensing solutions from Silicon Labs, visit www.silabs.com/pr/QuickSense and www.silabs.com/pr/lowpower.

Era of Connected Intelligence: Standardizing Healthcare Device Connectivity

by Cuauhtemoc Medina Rimoldi, *Freescale Semiconductor, Inc.*

Advances in electronic monitoring are revolutionizing healthcare, but the real benefits won't arrive until these new devices can be seamlessly interconnected.

Can you imagine a world without mobile phones, the Internet, or laptops? How would our world be different?

Billions of emails are sent daily. Hundreds of thousands of computers are used every day. How many mobile phones do you have at home? To make things even more exciting, many devices which were not previously connected are being "plugged in": refrigerators, TVs, photo frames, ovens, washing machines, and healthcare devices.

What is the value of connecting healthcare devices?

You might ask yourself, "Why would we want to connect medical devices?" Picture an elderly person resting at home after years of hard work. Suddenly, that person receives an appointment reminder from their mobile phone indicating that they are to perform a routine health check. The person goes to their living room where a small, easy-to-use, portable blood pressure monitor assists in executing the test. The data is transmitted automatically to the preferred health repository where an algorithm is executed to evaluate the patient's status. There is no need to call the physician. In other cases, the automated diagnosis could set up a patient-physician video conference for further evaluation. As you can probably already imagine, the technology is available to do this now.

Imagine a situation where a different subject wants to lose weight over a predetermined period of time. The subject wears a coin-sized activity monitor to track how many steps are taken. The monitor then calculates how many calories are burned. Let's imagine that this person jogs for 30 minutes and then has lunch afterwards. The activity monitor has already transmitted the burned-calorie data to their mobile phone, which automatically links to a web application to process the data. The mobile phone serves as a display and suggests a varied menu to help our subject achieve the goal of losing weight.

The next example involves a diabetic patient. Our patient is listening to their favorite music on their mobile phone when the device interrupts to advise the patient to measure their glucose level. Our patient plugs in a small add-in which works with most standard mobile phone connectors and allows the patient to execute the test. This add-in is a portable glucose meter which uses the phone's graphic LCD to report measurements and uses its connectivity to update the preferred health repository. The physician can then track the progress of the treatment. The mobile phone accessory can also communicate to a wearable insulin delivery device to release the correct dose.

What are the challenges of healthcare device connectivity?

Technology businesses are very similar. Most standard technologies are sucessful in the consumer market and medical/healthcare devices are not an exception. Do you remember the battle over videotape formats in the 1970s and 1980s, or the recent audio player debate where a magnetic optical disc and flash memory were the contestants? What are the standard technologies for medical/healthcare device connectivity?

There are not very many major portable medical device manufacturers who can comply with the most exhaustive medical device regulations, such as the FDA. There is some standardization for medical device connectivity, but the standardization level is no different from that of general portable consumer applications, and in fact, medical device advancements are being driven by it.

The move toward standards has started to evolve with the creation of consortiums, consisting of medical device manufacturers, pharmaceutical companies, technology vendors, and service providers, whose aim is to develop standards for multi-vendor connectivity. This can help achieve consensus among suppliers and therefore create a whole new market of services and products and eventually change the way healthcare is delivered.

Selecting a standard

One of the strongest organizations in terms of medical device standardization is the Continua® Health Alliance (http://www.continuaalliance.org). This organization unites smart technology and medical devices with healthcare industry leaders to empower patients not only to exchange vital information, but to change the way they manage health and wellness. The Continua Health Alliance is a forum consisting of more than 230 member companies who have come together to promote multi-vendor device interoperability by forming workgroups to set standards for medical systems.

In order to build trust, this organization has created a product certification program with a recognizable logo which signifies interoperability with other certified products. Additionally, products that carry this logo may have a competitive advantage and may provide benefits such as being considered "easy-to-use," following Continua Health Alliance standards, or "future proofed" – meaning that they won't become prematurely obsolete. The products can be sold

through multiple channels, such as directly to consumers and through prescriptions written by healthcare providers, or sold by governments, health plans, or other service providers to their constituents.

Being part of such an organization has additional benefits, such as promotions in different channels. All of these products and services are creating a whole new market, and opportunities around medical devices will keep growing in the upcoming years.

Semiconductor vendors address medical connectivity needs

The semiconductor vendor was not traditionally involved at this level of the portable medical device application. Historically, the "secret sauce" of the application has been kept, and will continue to be kept, by the medical device manufacturer. It makes sense, as medical innovations are possible only after years of research, development, and comprehensive testing. However, connectivity is not the core competence of the medical device manufacturer. Therefore, there is an opportunity for the semiconductor vendor to provide key differentiations by offering a software suite capable of transmitting this data. One might think that common protocols, such as USB or Ethernet, could be used in medical applications. That is partially correct. The only difference from a conventional consumer application is having to deal with sensitive data that directly impacts diagnostics and therefore a person's health.

The medical device manufacturer needs to take advantage of existing communication protocols popular in consumer applications for two main reasons. The first is cost, which is associated with design cycles, service providers, and semiconductor technology availability. The second is associated with end-user acceptance and usability. Why would one want to transfer data through a custom adapter or protocol if at the end it is going to be analyzed by a physician or the sender on a tablet computer that has USB or Wi-Fi. Medical devices need to utilize the same protocol to communicate with consumer products and among themselves, but they still need to keep a formal data structure to preserve device interoperability and to correctly process and link related data. This is solved by adding an extra software layer between the communication protocol and the application.

Standards such as IEEE 11073-20601 (Optimized Exchange Protocol) are defined under Continua Health Alliance device interoperability guidelines. Microcontroller and microprocessor vendors such as Freescale Semiconductor offer software suites like the Freescale Medical Connectivity Library, which allows the medical device designer to apply various device specializations defined by the standard without worrying about the implementation details of the protocol (see Figure 2). The library is transport- and platform-independent, allowing the use of different transport technologies, such as Ethernet or USB, or platforms such as MQX (a complimentary RTOS from Freescale). Under these conditions, a USB-enabled glucometer could communicate with a manager application which transfers the data (under the same medical specialization protocol) through Ethernet to a health repository webpage. This manager application could be running on a tablet computer with Bluetooth® wireless technology or on a concentrator node in a hospital enabled with ZigBee® technology. Both of them could have connectivity to the Internet and communicate with each other independently.

Additionally, Freescale offers its customers a complimentary USB stack that features the Personal Healthcare Device Class (PHDC). The PHDC is a standard implementation of USB for medical devices. IEEE 11073 is also the foundation of the USB stack with PHDC. This provides structure to the communication interface by defining commands to access data, structuring data to be transmitted, and defining communication states. Figure 1 shows the proposed software architecture.

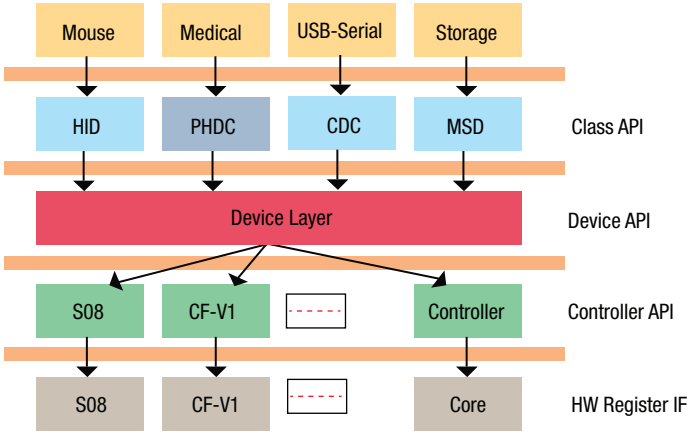


Figure 1: Freescale USB stack with PHDC support.

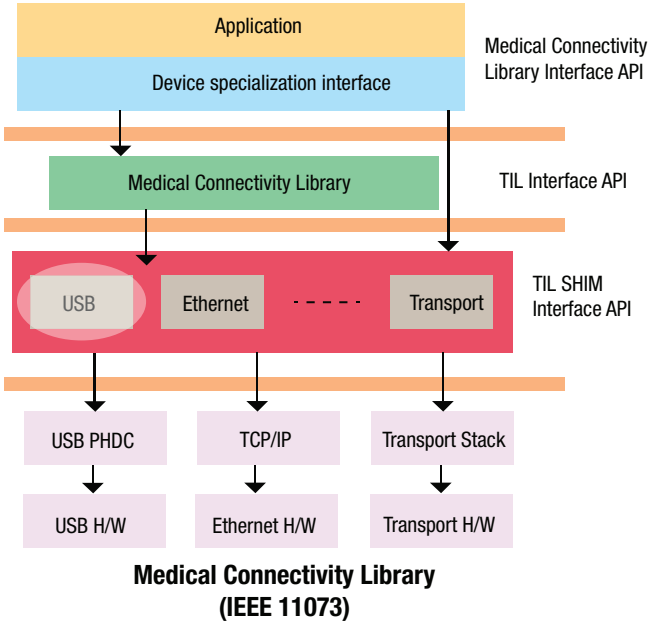


Figure 2: Freescale Medical Connectivity Library.

Freescale offers this ready-to-use software as part of the TWR-MCF51MM-KIT and TWR-S08MM128-KIT development board demo programs. The demo emulates a glucometer and a weight-scale device. With the push of a button, one can switch between the devices and the manager application (running on a PC) will recognize the device as a USB PHDC device (glucometer or weight scale). The manager application running on the PC transfers the data through Ethernet and uploads it to a well-known health repository, http://www.google.com/health. This proves the concept and enables users to develop applications that follow Continua Health Alliance guidelines. Freescale is not responsible for certifying the application.

The medical device vendor would need to apply for certification at Continua Health Alliance to ensure compliance, and to be allowed to use the consortium logo and additional benefits.

The USB stack with PHDC and the Medical Connectivity Library helps medical device vendors improve time-to-market and optimize budgets. The software suites have been ported to several 8-bit and 32-bit microcontroller devices and are planned to be ported to additional devices in the future. Freescale offers a wide range of microcontrollers with an integrated measurement engine (operational and transimpedance amplifiers, 12-bit DAC, 16-bit ADC and internal voltage reference). These devices are ideal for portable medical applications requiring signal instrumentation such as pulse oximeters, blood pressure monitors, and glucometers. Entry level devices are those in the 8-bit 9S08MM family, followed by the MCF51MM 32-bit ColdFire® V1 ultra-low-power microcontroller, and expanding to the recently announced high-performance ARM® Cortex™-M4 based Kinetis™ K50 microcontroller. These devices provide multiple connectivity options such as a USB device and the host or an Ethernet controller (see Figure 3).

Summary

Standardization is critical, especially as new markets emerge. Being a part of big alliances with important member companies is vital to ensure the interoperability and sustained development of personal medical devices. The Continua Health Alliance is a benchmark organization in the medical device industry by providing product certifications.

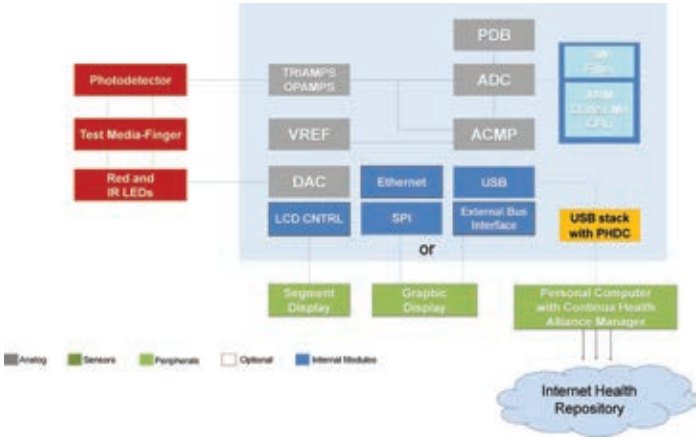
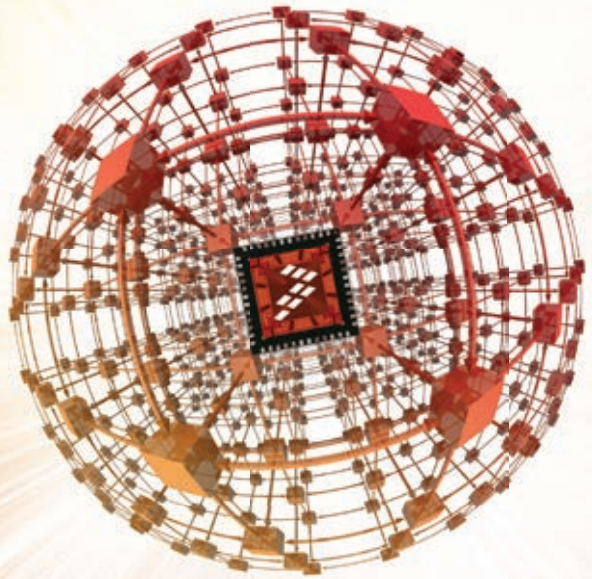


Figure 3: Freescale Kinetis K50 based pulse oximeter system diagram.

Semiconductor companies have started to differentiate themselves when approaching a medical device vendor. Companies like Freescale offer complimentary software suites to help reduce time-to-market and speed development. Freescale is a promoter member of the Continua Health Alliance and supports the advancement of the Continua vision. Microcontrollers like the 8-bit S08 9S08MM, the 32-bit ColdFire MCF51MM MCU, and the ARM Cortex-M4 based Kinetis K50 microcontroller integrate a measurement engine to enable future portable medical/healthcare devices.

Kinetis Microcontrollers

Design Potential. Realized.



- The first broad-market mixed signal MCU family based on the ARM® Cortex™-M4 core
- One of the most comprehensive ARM enablement portfolios in the industry
- Over 200 pin-, peripheral- and software-compatible MCUs with exceptional performance, memory and feature scalability
- Innovative 90 nm thin-film-storage flash technology with FlexMemory delivering high-endurance EEPROM
- Ultra-low-power performance and rich mixed-signal integration

Kinetis Family Positioning

K10 Family: 32 KB to 1 MB, ultra-low-power, advanced mixed signal, touch sensing, CAN
K20 Family: 32 KB to 1 MB, ultra-low-power, advanced mixed signal, touch sensing, CAN, USB OTG
K30 Family: 32 KB to 512 KB, ultra-low-power, advanced mixed signal, touch sensing, CAN, segment LCD
K40 Family: 32 KB to 512 KB, ultra-low-power, advanced mixed signal, touch sensing, CAN, USB OTG, segment LCD
K60 Family: 256 KB to 1 MB, ultra-low-power

freescale semiconductor

www.digikey.com/freescale-mcu

AVR32 ABDAC Audio Bitstream DAC Driver Example

contributed by Atmel

The ABDAC peripheral on AVR32 MCUs is quite suitable for generating audio playback. This article explains how to do it and includes a link to an example driver that generates a sine wave output.

Many embedded applications increasingly feature audio playback, whether simple audio feedback in response to user inputs or full high-speed streaming audio. By using the generic clock interface, the ABDAC on Atmel AVR32 MCUs is capable of supporting a wide range of playback frequencies.

Functional description

The ABDAC is a very simple peripheral and its use is straightforward. It needs a clock signal provided from the generic clock system, and data input to the channels. The block diagram in Figure 1 gives an overview of the module. For a detailed description of the ABDAC peripheral see the datasheet for the device.

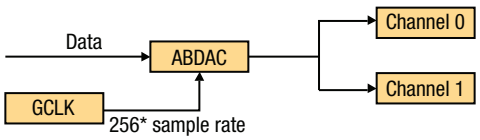


Figure 1: Clock and data path block diagram.

Generic clock

The ABDAC uses a generic clock to provide the sample frequency. This generic clock is hard wired inside the device and must be 256 times the sample frequency.

The generic clock should be configured and enabled before the ABDAC is enabled. For a description of which generic clock is used, see the clock section in the datasheet for the device. Further configurations of the generic clock are also described in this section.

The generic clock output range may be limited by its source clock frequency. It is therefore vital to design in an oscillator which can provide a base frequency that is dividable by the generic clock divider in order to reach the required output sample rate. Table 1 shows examples.

Channels

When the ABDAC is enabled, it expects the Sample Data Register (SDR) to be updated at the same interval as the output sample rate. Both channels can be updated with one written instruction, since they are in the same I/O register (SDR).

Table 1: Base frequencies needed for an output sample rate.

Output sample rates	OSC or PLL frequency	GCLK divider
48000 Hz, 240000 Hz, 12000 Hz	24.576 MHz	2, 4, 8
44100 Hz, 22050 Hz, 11025 Hz	22.5792 MHz	2, 4, 8
32000 Hz, 16000 Hz, 8000 Hz	16.384 MHz	2, 4, 8

If the sample data register is not updated within 256 clock cycles from the generic clock input to the ABDAC, the underrun bit will be set in the Interrupt Status Register (ISR). Underruns are a sign of too much CPU load, and therefore the application should be implemented by using interrupts, or even better, direct memory access (DMA) if available in the device.

Interrupts

There are two interrupts available to offload the CPU.

The TX_READY interrupt can be used as a trigger to signal that the next sample for each channel can be written.

The application should also enable the underrun interrupt to handle underruns when filling the Sample Data Register (SDR). Underruns will cause glitches and noise on the output signals.

If the underrun interrupt triggers, it is a sign of CPU overloading because the application was not able to provide the data in time.

DMA

The ABDAC may be connected to a DMA controller on the device. This will offload the CPU when transferring data from a buffer in RAM to the ABDAC. The application will only need to fill a buffer and pass the buffer address to the DMA controller.

Triggers for when a buffer is complete will let the application know when to pass a new buffer to the DMA controller.

The underrun interrupt is vital for DMA transfers, as it will indicate that the data busses in the device are overloaded or the DMA transfer to the ABDAC does not have enough priority.

Electrical connection

The output from the device is not intended for driving headphones or speakers. The pads limit the maximum amount of current.
In the majority of all practical cases, this will not be enough to drive a low-impedance source.

Because of this limitation, an external amplifier should be connected to the output lines to amplify these signals. This amplifier device could also be used to control the volume.

For testing purposes, a line in or microphone input on a sound system can be used to evaluate the output signal.

Passive filter

For connecting the ABDAC to high-impedance devices, such as line in on an amplifier, a passive filter should be added. Figure 2 shows an example schematic.

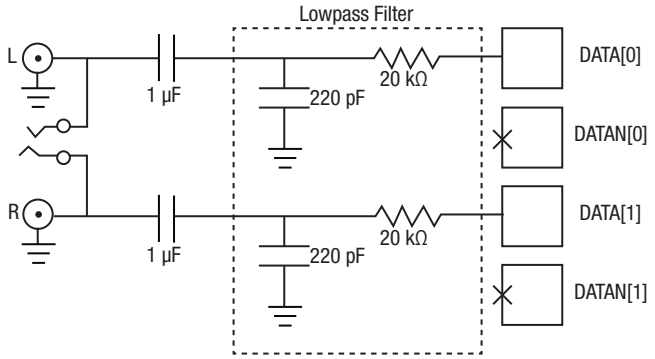


Figure 2: Line out with passive filter schematic.

External amplifier

An external amplifier is required if the ABDAC is driving low-impedance headphones or speakers directly. Figure 3 shows an example schematic using Texas Instruments' TPA152 stereo audio amplifier.

Driver implementation

Files

Full source code for the driver discussed in this article can be found at www.atmel.com/dyn/resources/prod_documents/AVR32120.zip.

The driver consists of two files, "dac.c" and "dac.h," where "dac.h" declares all functions and "dac.c" contains the source code. The only change needed in the driver is to specify the target device. The target device is specified at the top in "dac.h."

Example code

The example code for the driver outputs a sinus wave on both DAC channels. This output is enabled by a user input on a GPIO line. The wiring information is included in the documentation which accompanies the source code.

The example code is targeted for the ATSTK1000, but can, with some tweaking, work with any AVR32 devices with an ABDAC.

Figure 4 shows the flow of the example application. The application is implemented by polled function calls to make it less dependent on other modules.

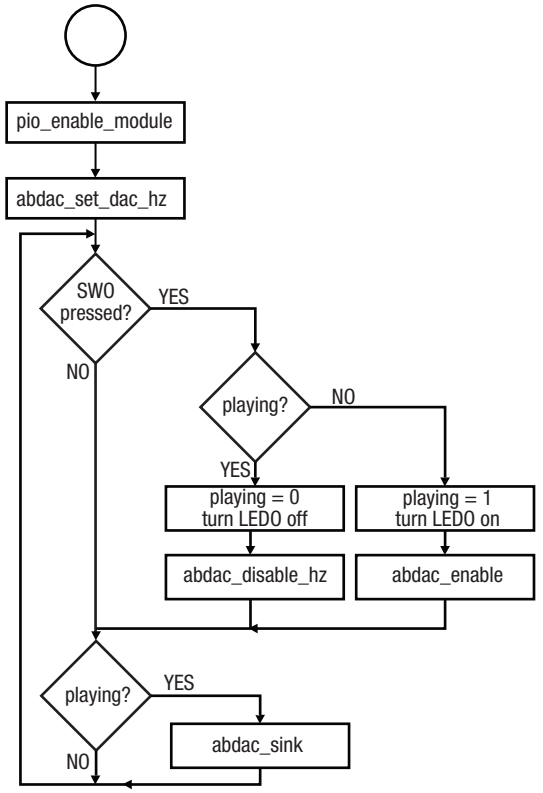


Figure 4: ABDAC example application flowchart.

Further reading

DMA controller

The DMA controller can offload the CPU while outputting data from the ABDAC. For more information about the DMA controller, see the appropriate chapter in the datasheet for the AVR32 device.

Interrupt

The ABDAC interface has an interrupt line connected to the interrupt controller (IC). Handling the ABDAC interrupt requires programming the IC before configuring the ABDAC. For more information and details about the interrupt controller, see the application note AVR32101: The AVR32 Interrupt Controller.

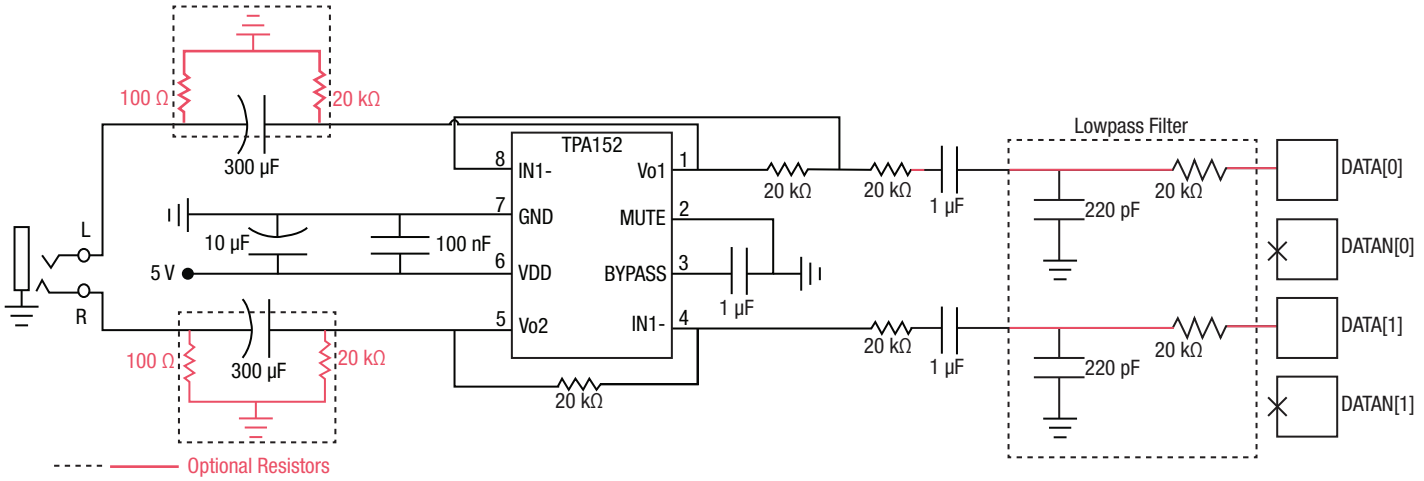


Figure 3: High power output with external amplifier schematic.

ReferenceDesignLibrary

New Design Database Gives Digi-Key Customers a Competitive Edge

Digi-Key's Reference Design Library (beta version) is a new online tool for electronic design engineers.

This interactive web-based library consists of proven designs contributed by reliable sources and a filtering capability based on each design's measured performance.

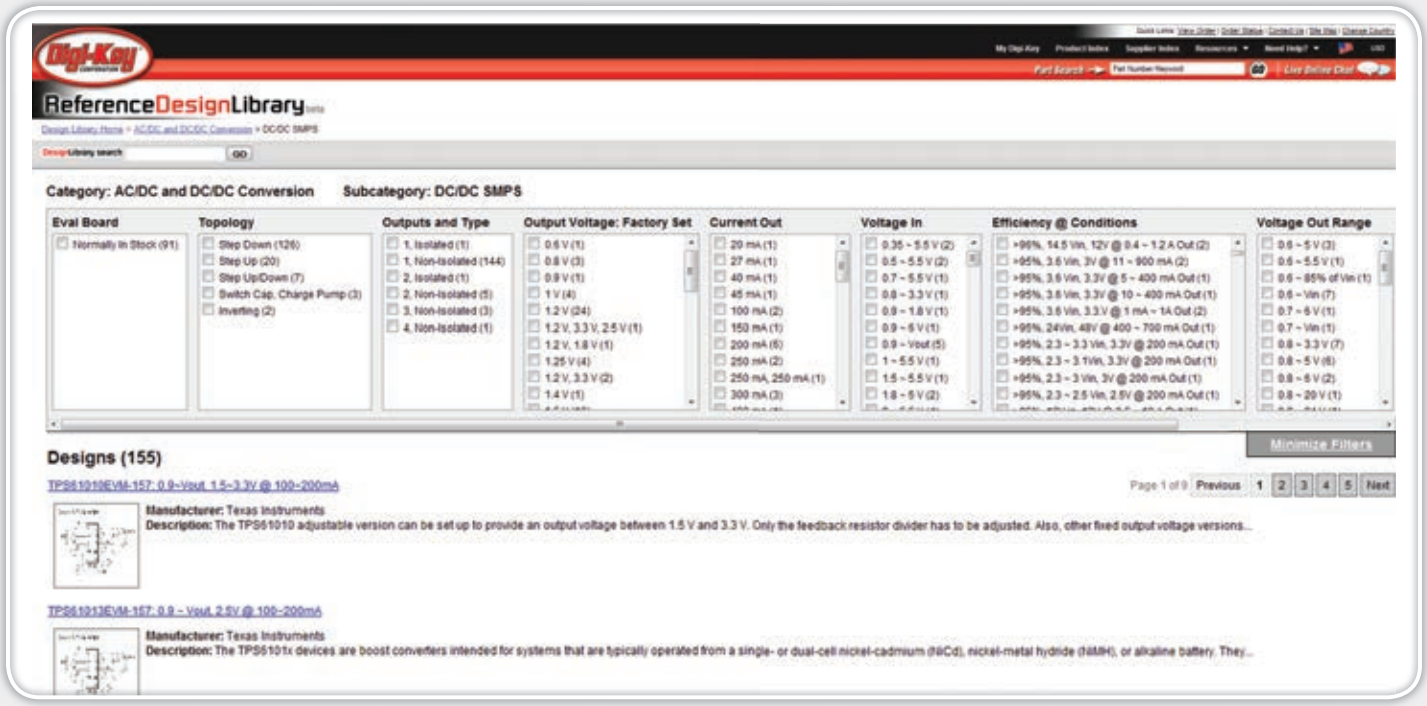
Digi-Key's Reference Design Library was created for the convenience of engineers and electronics designers. Creators of this resource understood the need for valuable, proven design blueprints from leading component suppliers to be available in one organized, searchable repository.

By clicking into categories, users of the Reference Design Library can search for designs based on voltage in, current out, outputs and types, and efficiency in certain conditions.

Digi-Key's Reference Design Library currently houses 400+ designs regarding AC/DC and DC/DC conversion, audio amplifiers, motor control, power management, sensor solutions, wireless communication, and lighting.

Instead of searching countless manufacturer websites and contacting companies to find needed designs, users can browse the Reference Design Library based upon category and subcategory. Currently, the lighting and AC/DC conversion categories are the most expansive, but new designs are frequently added.

This addition to Digi-Key's website is located on the homepage at **www.digikey.com** under the "Resources" section or by visiting **www.digikey.com/referencedesign**. Visit Digi-Key's Reference Design Library often to view the newest design additions.



www.digikey.com/referencedesign

Low-Power, Long Range, ISM Wireless Measuring Node

contributed by Analog Devices, Inc.

A low-power wireless sensor node needn't be either short range or complicated. This article describes a simple, inexpensive two-chip solution.

Ideally, a wireless measurement node is low power, has good range, and is easily interfaced to different sensors. Through the combination of three Analog Devices, Inc. parts, an intelligent measurement node with an average current consumption of <70 µA, a range of almost 1 km (in free space), and a data rate of one transmission/minute can be achieved, while also maintaining a 16-bit ADC performance (see Figure 1). This makes the circuit suitable for battery power and such applications as automation and remote sensing.

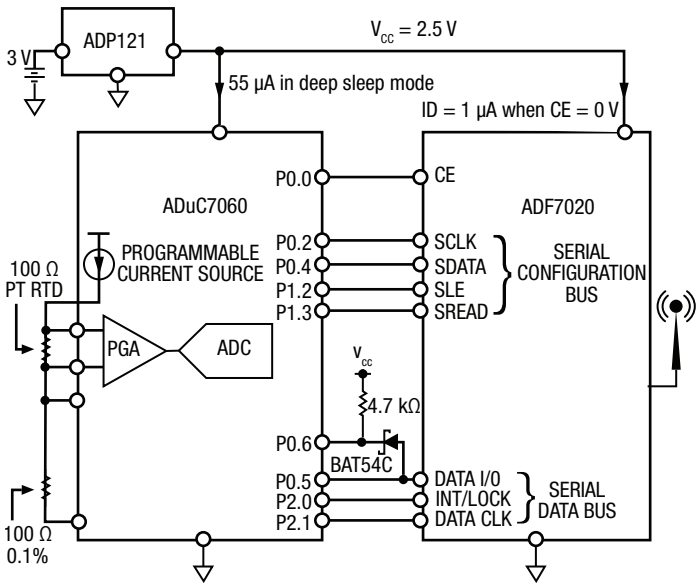


Figure 1: Low-power, long range, ISM wireless measurement node (simplified schematic: all connections and decoupling not shown).

The ADuC7060 precision analog microcontroller has a low-power ARM7 core as well as a myriad of precision analog functions. The onboard multiplexer, digitally programmable gain amplifier (PGA), voltage reference, programmable current sources, and 24-bit sigma-delta ADC allow almost any temperature and bridge sensors to be directly connected. In this case, a four-wire Pt100 (100 Ω platinum RTD) temperature sensor was chosen. Further details on the measuring circuit can be found in the AN-0970 Application Note.

The system consists of a low-power temperature measurement node that wakes once a minute, measures temperature, transmits this measurement at 10 kbps to the base node, and returns to sleep. The base node continuously listens for a package from the measurement node and sends this information to the PC via the UART for display in HyperTerminal.

The wireless band chosen for this application is the sub-GHz, license-free ISM (industrial, scientific, medical) band. The ADF7020 transceiver, which supports bands in the 431 MHz to 478 MHz frequency range as well as the 862 MHz to 956 MHz frequency range, is therefore a natural choice. This low-power transceiver requires very few external components, is easily connected to the ADuC7060 precision analog microcontroller, and offers excellent performance.

The ADP121 voltage regulator provides the 2.5 V supply from two 1.5 V batteries. The very low quiescent current of this voltage regulator (11 µA at no load) is paramount in maximizing battery lifetime.

Circuit description

Two buses connect the ADF7020 ISM transceiver with the ADuC7060 precision microcontroller. Both buses are serial and bidirectional. One of these buses configures the transceiver and requires four microprocessor ports. The second bus is the data bus, which enables the data transaction between controller and transceiver. This bus requires at least three microprocessor ports. In this particular application, two ports are used instead of one bidirectional port with two interrupts. This simplifies the software, but necessitates the use of an extra diode and resistor to separate incoming and outgoing data streams. A parallel combination of two Schottky diodes ensures a logic low, which is less than 200 mV. The BAT54C has two diodes in the same package (connecting Pin 1 and Pin 2 together for a parallel configuration). All digital ports on the ADuC7060 have programmable pull-up resistors; however, an external pull-up resistor is also required. With a data rate of 10 kbps, a 4.7 kΩ resistor works well.

Three factors determine the overall current drawn by the circuit: the requirement of the individual components in both sleep and active modes, the amount of time the system is active, and the amount of time the transceiver itself is active.

The first factor is addressed by choosing low-power components such as the ADuC7060 and the ADF7020. The second factor, minimizing the activity of the system, is achieved by keeping the system inactive as long as possible. It is worth considering the tradeoff between integer versus floating point arithmetic – in many cases, integer is sufficient,

has a shorter execution time, and thus provides greater savings. The final factor, reducing air time, is achieved in part by using a protocol with minimum overhead, but also, to a greater extent, by using the ADF7020, which has very high receiver sensitivity and good out-of-band rejection, thus maximizing the probability that the data package contains correct data.

Code description – general

The system spends the majority of time in deep sleep mode, with a current consumption of 50 µA to 60 µA (depending on ambient temperature). Timer 2 wakes the system every second. Every 60 seconds, an ADC measurement is executed, linearized, and transmitted. Timer 2 can wake the system from deep sleep; the other three timers cannot. Timer 2 is 16-bit, meaning that it wakes every second when running from a 32 kHz clock (in sleep mode). After the ADC is started, the system goes into pause mode (see the ADuC7060 data sheet for more information). This is a reduced power mode, albeit not as reduced as deep sleep. The ADC wakes the system when finished. A temperature value is calculated from the ADC results and is packaged and transmitted.

Packaging essentially means placing appropriate data in a buffer. In this case, the data consists of a 4-byte floating point temperature value and a 2-byte CRC (cyclic redundancy check). In a more complex system, a header with node address, received signal strength, and other information precedes this data. Before sending this buffer to the ADF7020 transceiver, an 8-byte preamble to help synchronize the receiving node and a 3-byte synchronization word, or sync word, are sent. This is a unique 3-byte number that is checked for a match at the receiver node before a package can be received.

The hardware is very similar on the receiving side; an ADF7020 transceiver is configured to listen for the unique sync word. After the sync word is received, the data package follows. The data is sent to the PC via the UART.

Flowcharts for the main loops of both the measurement node and the base receiving node are displayed in Figure 2.

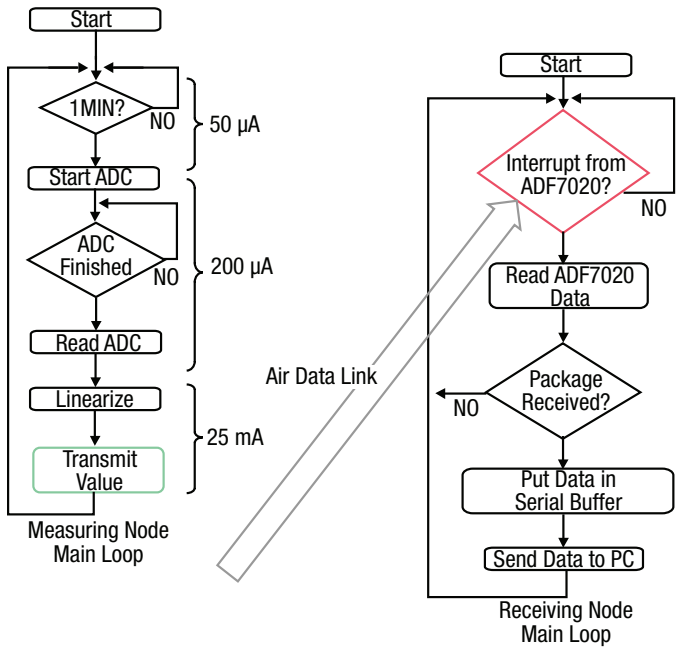


Figure 2: Measuring and receiving node main loop flowcharts.

Code description – ADF7020 driver

There are many modulation schemes supported by the ADF7020. In this case, the GFSK (Gaussian Frequency Shift Keying) is used. This has the benefit of having very good spectral efficiency. In this mode, the ADF7020 generates the data clock both when transmitting and receiving. The rising edge of this clock (DATA CLK) generates an interrupt, which causes the ADuC7060 to place the data on the output port bit-by-bit, as shown in Figure 3. When all the data has been clocked-out, the chip select is deasserted, and the ADuC7060 reenters deep sleep mode.

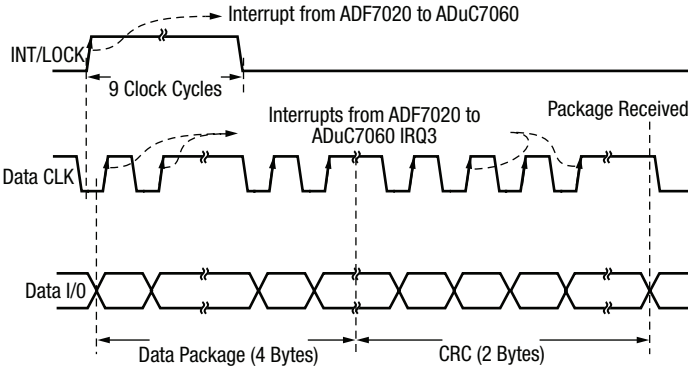


Figure 3: Data I/O timing.

On the receiving side, the ADF7020 generates an interrupt when a matching sync word is received (Port INT/LOCK goes high for nine clock cycles). This informs the ADuC7060 processor to prepare for the reception of a package. Each bit that is received from the package causes an interrupt in the ADuC7060. In the interrupt service routine (ISR), the bit stream is read and stored in a buffer. When all the bytes in the package have been received, a flag is set to indicate that a new package has been received. The main loop can now ensure the validity of the package by the checksum. A correct and complete package can be processed. In this case, this information is sent via the UART to the PC for display. The same ISR handles both the sending and receiving of data to/from the ADF7020 transceiver, as shown in Figure 4.

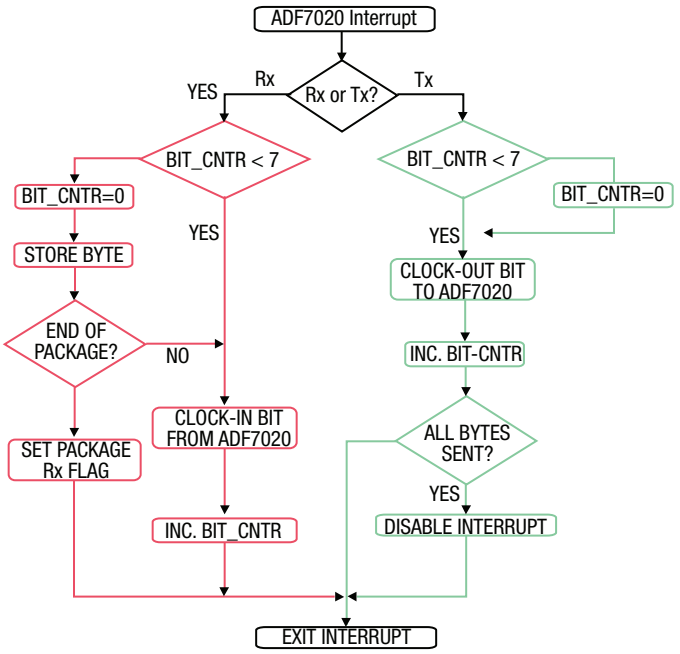


Figure 4: Interrupt service routines for handling Rx and Tx data.

Common variations

Depending on the desired frequency, there are a number of other products that can be used instead of the ADF7020. For example, for the 2.4 GHz frequency band, the ADF7242 is a very good choice.

LEARN MORE

Looney, Mike. AN-0970 Application Note. RTD Interfacing and Linearization Using an ADuC706x Microcontroller, Analog Devices. http://www.analog.com/static/imported-files/application_notes/AN-0970.pdf

Data sheets and evaluation boards

ADF7020 Data Sheet

ADF7020 Evaluation Board

ADF7020 Device Drivers

ADuC7060 Data Sheet

ADuC7060 Evaluation System

ADP121 Data Sheet

ARM7 Core and Microconverters

Analog Devices, Inc.

ARM7 is a powerful 32-bit RISC (reduced instruction set computer) microprocessor architecture from Advanced RISC Machines Ltd. (ARM). These core processors can be utilized in embedded applications such as cell phones, PDAs, and more.

The PTM provides a review of the device's features, a discussion of the product road map, and examples of I/Os and peripherals on ADI's MicroConverters.

ANALOG DEVICES

PTM PRODUCT TRAINING MODULES *Online...On Demand*®

www.digikey.com/ptm

PICDEM™ Lab Development Kit (DM163035)

The PICDEM™ Lab Development Kit is designed to provide a comprehensive development and learning platform for Microchip's Flash-based 6/8/14/18 and 20-pin 8-bit PIC® microcontrollers (MCUs). This kit is geared toward first-time PIC microcontroller users and students, and includes five of our most popular 8-bit PIC MCUs and a host of discrete components to create instructive applications. Expansion headers provide complete access/connectivity to all pins on the connected PIC MCUs and all mounted components.

A solderless prototyping block is included for quick exploration of the application examples described in the "hands-on" labs included in the user's guide. These labs provide an intuitive introduction to using common peripherals and include useful application examples, from lighting an LED to some basic mixed signal applications using the free HI-TECH C® PRO for the PIC10/12/16 MCU Family Lite Mode Compiler. This kit also includes Microchip's PICkit™ 2 Programmer/Debugger and a suite of free software tools that enable original applications to be developed quickly

Kit Contents:

- PICDEM Lab Development Board
- Component kit
- PICkit 2 Programmer/Debugger
- CD containing a comprehensive user guide, labs, application examples and tutorials

MICROCHIP

www.digikey.com/microchip-mcu

Real-Time: Some Notes on Microcontroller Interrupt Latency

by Jim Harrison, *Electronic Products*

Interrupts take a lot out of a high-speed processor, especially one that is heavily pipelined and, capable of issuing more than one instruction per cycle. There could be eight to ten instructions in flight at any one time that either have to be run to completion, or annulled and restarted once normal execution resumes.

The electrical engineer needs to check that the interrupt responds fast enough for the application and, that the overhead of the interrupt does not swamp the main application.

Just how fast can a given MCU perform an interrupt? That is certainly affected by the application, but it seems unreasonably hard to find a number for this item.

When an interrupt occurs, the CPU saves some of its registers and executes the interrupt service routine (ISR), and then returns to the highest-priority task in the ready state. Interrupts are usually maskable and nestable.

Just to be clear, latency is usually specified as the time between the interrupt request and execution of the first instruction in the interrupt service routine. However the "real latency" must include some housekeeping that must be done in the ISR, which can cause confusion.

The value in which an electrical engineer is usually interested is the worst-case interrupt latency. This is a sum of many different smaller delays.

1. The interrupt request signal needs to be synchronized to the CPU clock. Depending on the synchronization logic, typically up to three CPU cycles can be lost before the interrupt request has reached the CPU core.
2. The CPU will typically complete the current instruction. This instruction can take a lot of cycles, with divide, push-multiple, or memory-copy instructions requiring most clock cycles taking the most time. There are often additional cycles required for memory access. In an ARM7 system, for example, the instruction STMDB SP!,{R0-R11,LR} (Push parameters and perm.) Registers is typically the worst case instruction. It stores 13 32-bit registers on the stack and requires 15 clock cycles.
3. The memory system may require additional cycles for wait states.
4. After completion of the current instruction, the CPU performs a mode switch or pushes registers (typically PC and flag registers) on the stack. In general, modern CPUs (such as ARM) perform a mode switch, which requires less CPU cycles than saving registers.
5. If your CPU is pipelined, the mode switch has flushed the pipeline and a few more cycles are required to refill it. But we are not done yet. In more complex systems, there can be additional causes for interrupt latencies.

In more complex systems, there can be additional cause for interrupt latencies.

1. Latencies cause by cache line fill:
   If the memory system has one or multiple caches, these may not contain the required data. Then, not only the required data is loaded from memory, but in many cases a complete line fill needs to be performed, reading multiple words from memory.
2. Latencies caused by cache write back:
   A cache miss may cause a line to replaced. If this line is marked as dirty, it needs to be written back to main memory, causing an additional delay.
3. Latencies caused by Memory Management Units (MMU) translation table walks:
   Translation table walks can take a considerable amount of time, especially as they involve potentially slow main memory accesses. In real-time interrupt handlers, translation table walks caused by the Translation Lookaside Buffer (TLB) not containing translations for the handler and/or the data it accesses can increase interrupt latency significantly.
4. Latencies caused by the application program:
   The application program can cause additional latencies by disabling interrupts.
5. Latencies caused by interrupt routines:
   If the application has more than one urgent interrupt, they cannot be masked off so another may be requested, lengthening the total time.
6. Latencies caused by the RTOS:
   A RTOS also needs to temporarily disable the interrupts which can call API-functions. Some RTOSs disable all interrupts, effectively worsening interrupt latencies for all interrupts, some (like embOS from Segger) disable only low-priority interrupts.

ARM7 and ARM Cortex

The ARM7 and ARM Cortex are very different in the interrupt area. By integrating the interrupt controller in the processor, Cortex-M3 processor-based microcontrollers have one interrupt vector entry and interrupt handler per interrupt source. This avoids the need for re-entrant interrupt handlers, which have a negative effect on interrupt latency.

	ARM7TDMI	Cortex-M3
Interrupt controller	External to processor	Integrated nested vectored interrupt controller
Interrupt handlers	One fast (nFIQ) and one slow (nIRQ)	One handler per interrupt source
RTOS system timer	Uses one timer of the microcontroller	Uses integrated "SysTick" timer on the processor
System calls	SWI instruction (interrupts disabled)	SVC instruction (interrupts enabled)
Memory interface	Single interface, data read/write takes 3 cycles	Separate instruction and data bus interfaces, single cycle data read/write
Pipeline	Three-stage	Three-stage with branch speculation
Bit manipulation	Read, modify, write	Single instruction

The Cortex-M3 also accelerates the execution of interrupt handlers with logic to automatically save its general purpose and status registers in the stack when an interrupt arrives. The M3 is made even more efficient, in certain circumstances, by tail-chaining interrupts that arrive at the same time, as shown in Figure 1.

The interrupt latency is up to 12 cycles for the Cortex-M3 processor-based MCU, and the context switch time is <4 µs, while the ARM7 is <7 µs.

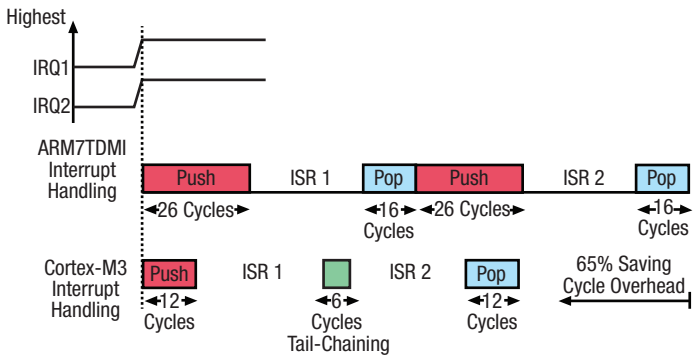


Figure 1: Tail-chaining on Cortex-M3 processor speeds up things.

Microchip

According to Keith Curtis, technical staff engineer at Microchip, the 8-bit PIC-16/PIC-18 MCUs take 12 to 20 clock cycles to get to the ISR – depending on the type of instruction that was in progress at interrupt time. Then, in the ISR, the compiler will add instructions to determine where the interrupt originated and to push some registers. If you are using assembly language, you would put in your own items that need pushing, perhaps none.

Microchip's 32-bit PIC32 MCUs, according to Adrian Aur, applications engineer, will take a maximum of 11 clock cycles to get to the ISR where you will save at least some registers – worst case, all 32 of them need one clock cycle each. If you are responding to INT7, the highest priority (and not interruptible), a set of shadow registers will be used, making response much faster. Then, the RTOS may want to make a thread change, or enable nested interrupts when running at lower priority levels, which will add some latency. Other than that, you should be fine.

Atmel

In 2008, Electronic Products Magazine gave Atmel a Product of the Year Award for the AVR XMEGA microcontroller family. The biggest reason for that was its innovative eight-channel event system which enables inter-peripheral communication without CPU or DMA usage using a bus separate from the data bus. The benefit of this is predictable, low-latency, inter-peripheral signal communication, reduced CPU usage, and the freeing of interrupt resources.

Independent of the CPU and DMA, the response time for the event system will never be more than two clock cycles of the I/O clock (usually 62.5 ns).

The XMEGA uses a Harvard architecture with the program memory separate from data. Program memory is accessed with single level pipelining. While one instruction is being executed, the next is prefetched. Performance is enhanced with the fast-access RISC register file – 32 x 8-bit general-purpose working registers. Within one single clock cycle, XMEGA can feed two arbitrary registers from the register file to the ALU, do a requested operation, and write back the result to an arbitrary register.

The interrupt response time for all the enabled interrupts is a minimum of five CPU clock cycles. During these five clock cycles, the program counter is pushed on the stack. After five clock cycles, the program vector for the interrupt is executed. The jump to the interrupt handler takes three clock cycles.

If an interrupt occurs during execution of a multicycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the device is in sleep mode, the interrupt execution response time is increased by five clock cycles. In addition, the response time is increased by the start-up time from the selected sleep mode.

A return from an interrupt-handling routine takes five clock cycles. During these five clock cycles, the program counter is popped from the stack and the stack pointer is incremented.

PIC XLP Development Board Demonstrates How Low You Can Go

by John Donovan, *Low-Power Design*

A wide range of sleep modes enables applications to idle at power levels so low they are hard to measure.

Historically, microcontroller vendor competition was largely dominated by a "faster is better" philosophy. Having finally hit the power wall – in spite of the combined efforts of Moore's Law and Ohm's Law – semiconductor manufacturers are now competing to see who can make the lowest power MCUs. "Low power" has now been replaced by "ultra-low-power." When it comes to low power, the race to the bottom is heating up.

Embedded developers currently have a wide range of very capable low-power MCUs from which to choose. It is an embarrassment of riches, compounded by conflicting (if carefully qualified) claims that MCUs have the lowest active, standby, sleep, deep sleep, or other operating characteristics. In fact, the proliferation of process technologies, power management techniques, and operating modes has made it difficult to do an apples to apples comparison between different MCUs just by looking at their data sheets.

Embedded developers are increasingly utilizing evaluation and development boards to test their application code on different MCU platforms. This article will take a hands-on look at Microchip's XLP 16-bit Development Board (see Figure 1), which is designed to showcase the numerous software-selectable, low-power operating modes for its nanoWatt XLP eXtreme Low Power MCUs – in this case the 16-bit PIC24F16KA102. The board and its demonstration program – accompanied by well commented C source code – makes it easy to explore both the parameters and the trade-offs involved in utilizing various power-down modes.

Powering down

Active or dynamic power is largely the power required to switch logic gates. As processors have become more complex, moving to smaller process geometries has continually reduced core voltages, with the resulting power savings far outweighing the added current load of the extra transistors. Thanks to Ohm's Law, core voltage (V_{DDCORE}) has a dramatic impact on both active and static power. Still, with Moore's Law now running headlong into some laws of physics, that free ride is coming to an end.

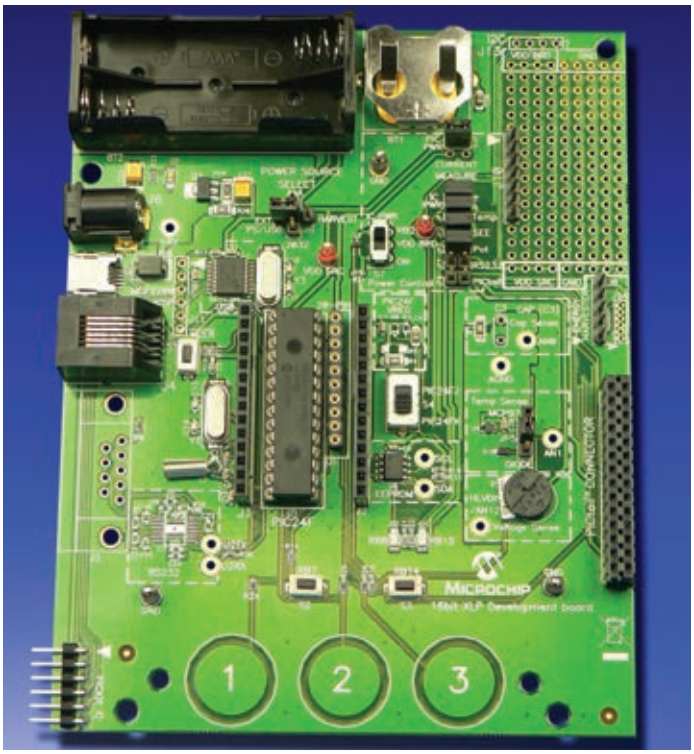


Figure 1: Microchip's 16-bit XLP Development Board (Source: Microchip. Used with permission).

Active power also correlates directly with clock speed, so most MCUs will automatically dial back the system and/or secondary clocks whenever the application allows it. Dynamic voltage and frequency scaling – once the domain of PMICs – are increasingly common in higher-end MCUs. Combine that with multiple fine-grain voltage islands, numerous power-down states with quick recovery times, and so-called smart peripherals, and it becomes difficult to see how much more MCU designers can do to reduce active power.

Static power is another matter, and that is where the greatest gains are being made. Static power is what you are left with when the system clock is turned off. It's largely dependent on process technologies and device design to control transistor leakage, which also varies with voltage and temperature. Within those design limitations, MCUs that can reduce V_{DDCORE} as far and as often as possible will have the greatest impact on both static and dynamic power. That is basically what Microchip's "nanoWatt Technology" is all about.

NanoWatt sleep modes

Microchip introduced its nanoWatt Technology for PIC MCUs back in 2003. It was not a new technology so much as a series of process, design, and software-controlled power saving features which enable these MCUs to deliver "overall power consumption in the nanoWatt range while in Sleep mode," according to the data sheet. New power saving features included an idle mode, an on-chip, high-speed oscillator with PLL and programmable prescaler, a watchdog timer (WDT) with an extended time-out interval, ultra-low-power wakeup (ULPWU), low-power options for the system and secondary oscillators, and a low-power, software-controlled brown-out reset (BOR).

Raising (or lowering) the bar, Microchip's more recent nanoWatt XLP Technology defines this specification as any MCU below:

- 100 nA for power-down current (I_{PD})
- 800 nA for watchdog timer current (I_{WDT})
- 800 nA for real-time clock and calendar (I_{RTCC})

The key to XLP is the use of several software-selectable hardware configurations which enable an application to dynamically change power consumption during execution. Instead of your compiler making these decisions, your application can make them at runtime. Table 1 details what happens during the various operating modes.

Table 1: Power-saving modes for nanoWatt Technology devices (Source: Microchip. Used with permission).

Operating Mode	Active Clocks	Active Peripherals	Wake-up Sources	Typical Current	Typical Usage
Deep Sleep (1)	• Timer1/SOSC • INTRC/LPRC	• RTCC • DSWDT • DSBOR • INT0	• RTCC • DSWDT • DSBOR • INT0 • MCLR	< 5 Na	• Long life, battery-based applications • Applications with increased Sleep times
Sleep	• Timer1/SOSC • INTRC/LPRC • A/D RC	• RTCC • WDT • ADC • Comparators • CVREF • INTx • Timer1 • HLVD • BOR	All device wake-up sources (see device data sheet)	50-100 nA	Most low-power applications
Idle	• Timer1/SOSC • INTRC/LPRC • A/D RC	All Peripherals	All device wake-up sources (see device data sheet)	25% of Run Current	Any time the device is waiting for an event to occur (e.g., external or peripheral interrupts)
Doze(2)	All Clocks	All Peripherals	Software or interrupt wake-up	35-75% of Run Current	Applications with high-speed peripherals, but requiring low CPU use
Run	All Clocks	All Peripherals	N/A	See device data sheet	Normal operation

Note: 1: Available on PIC18 and PIC24 with nanoWatt XLP™ Technology only.
2: Available on PIC24, dsPIC and PIC32 devices only.

Run, Idle, and Doze modes should be pretty clear from Table 1. However, take a brief look at the Sleep Modes before we do a reality check using the XLP Development Board.

- **Sleep Mode** – This is the basic low-power mode for all PIC MCUs. Here the system clock and most peripheral clocks are shut down, with state retention for the program counter (PC), SFRs, and anything in RAM. Wake up can be initiated by the WDT, the secondary clock, or various external interrupt sources. Wake up times vary but are usually in the low microseconds.
- **Deep Sleep** – In Deep Sleep mode, the MCU core, most peripherals, on-chip voltage regulators, and (sometimes) RAM are powered down. While Deep Sleep is the lowest power state, waking up from it is non-trivial (see Figure 2), requiring a device reset rather than the smooth transition back from Sleep Mode. However,in going into Deep Sleep mode all I/O states, the secondary clock and RTCC are maintained so that the rest of the system can continue operating. Upon reset the program resumes execution from the reset vector once the application reconfigures the peripherals and I/O registers.

Because of the overhead (read:time) required to implement Deep Sleep Mode, embedded developers need to consider when to utilize it. Microchip recommends using it when your application can support sleep times of one second or longer, when it doesn't require access to any peripherals while asleep, when it still needs accurate, low-power timekeeping, and when it's operating in environments that may experience extreme temperatures.

Deep Sleep makes a lot of sense for low-power wireless sensor nodes that may need to operate for years from a coin cell battery and only wake up every few seconds to transmit data for a few microseconds. It also makes sense in energy scavenging applications, an option for the XLP board but not one I will to review here. In less extreme applications, standard Sleep Mode is recommended.

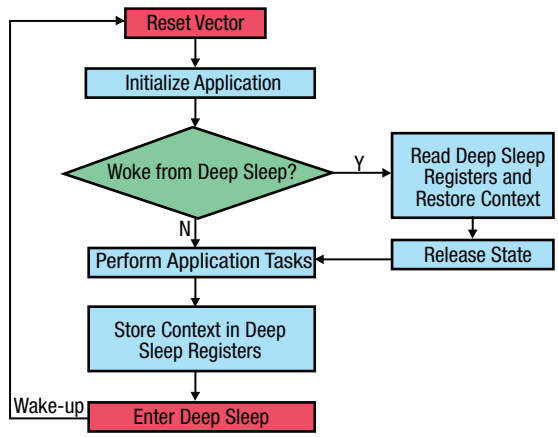


Figure 2: Procedure for waking up from Deep Sleep Mode (Source: Microchip. Used with permission).

XLP – the demo

Microchip chose to build its XLP Development Board around the PIC24F16KA102 in order to demonstrate that chip's low-power capabilities, which are broken out in Table 2. Without going too far into the data sheet details, the chip features a 16-bit, 16 MIPS core running at 1.8 to 3.6 V, 16 kbytes of flash, 16 kbytes of SRAM, 1.5 kbytes of DRAM, 512 bytes of data EEPROM, a 10-bit ADC, two UARTs, and a charge time measurement unit (CTMU) enabling nine channels of capacitive touch (see Figure 3).

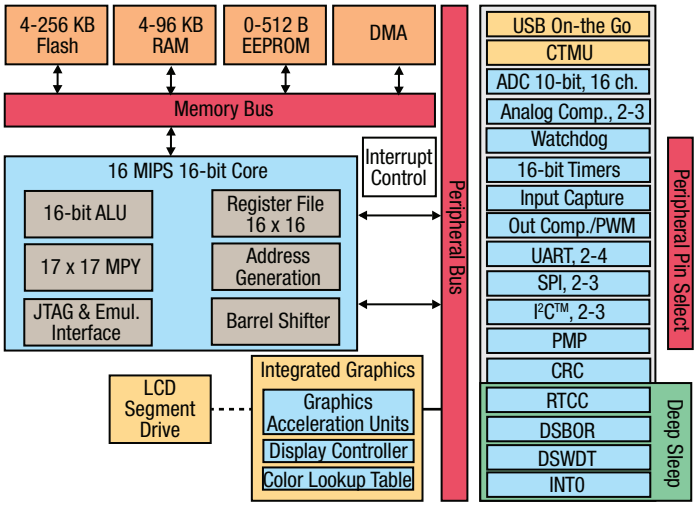


Figure 3: PIC24F16 family block diagram (Source: Microchip. Used with permission).

Table 2: PIC24F16KA102 power specifications.

Deep Sleep (nA)	20
Sleep (nA)	25
WDT (nA)	420
32-kHz Oscillator/RTCC (nA)	520
I/O port leakage (nA)	±50
1 MHz Run (µA)	195
Minimum V_{DD}	1.8

The board itself looks to be plug-and-play, since you only need to plug in a USB cable to power it and watch the sample program run that is already embedded in the MCU. In practice, there are a lot of moving parts to download, configure, and figure out before you can begin.

Getting started

The kit itself contains the board, a mini-B USB cable, and a brief information sheet. The board itself can run from two AAA batteries, a CR2032 coin cell, a Cymbet CBC-EVAL-08 energy scavenging demo board, a USB cable, or an external 9 V power supply – none of which are supplied. I chose to power it from the USB cable.

First you need to download and install the XLP 16-bit Development Board code (currently v1.2) from Microchip's web site. The software only claims to run on Windows XP and Vista, but it installed easily on my 64-bit Windows 7 machine, though as a 32-bit program.
The board comes with a demo program already installed in flash, so you only need to run a terminal emulation program on your PC to observe it in operation – which proved to be a lot more easily said than done.

Windows XP and Vista come with HyperTerminal, which cannot support the high data speed required by the XLP board – an amazing 1 Mbyte/second. A Microchip engineer explained that speed is necessary for fast data exchange with the Cymbet energy harvesting board, whose tiny thin-film batteries need to push out data quickly before they fade out. The XLP software download includes the open source program RealTerm, which installs along with the demo program source code and documentation.

After a smooth install, I connected the board, turned on RealTerm – and couldn't get it to work. After reading lots of release notes, checking ports with Device Manager, and changing parameters, I finally found a directory called USB Serial Emulator, a subdirectory called "drivers", and finally USBDriverInstaller.exe. Running that program solved the problem (see Figure 4). It still eludes me why you need this driver when there is a Microchip MCP2200 USB-to-UART serial converter sitting right next to the USB jack.

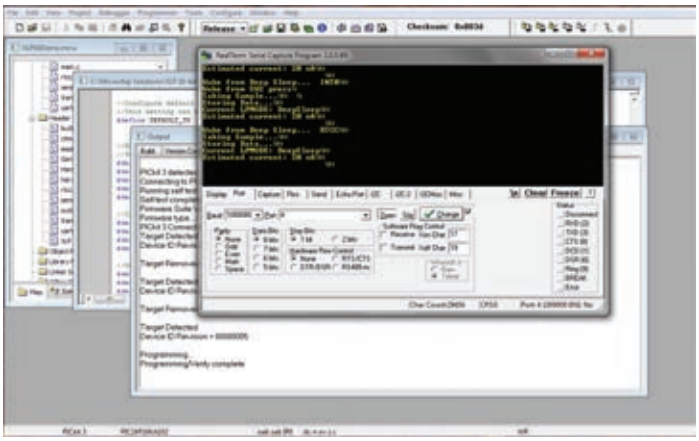


Figure 4: RealTerm running the XLP demo program.

The XLP board incorporates sensors for temperature, voltage, and touch, all of which you can switch in or out using jumpers, or you can poll or ignore them using software. Pressing S3 for more than two seconds toggles the MCU between Sleep, Deep Sleep, and Idle Modes; pressing it briefly disconnects the UART so you can measure the system power with it out of action. Pressing S2 cycles between the temperature sensor, voltage sensor, and capacitive touch pads, with the results from all three measurements being shown instantly. S1 (MCLR Reset) can wake the MCU from Deep Sleep Mode, as can S2 (INT0) and the RTCC, which wakes the MCU every 10 seconds in the demo program.

Measuring nanoamps

Not that I do not trust Microchip, but both the fun and much of the point of doing a review involves checking out their claims that this is an ultra-low-power chip. Up until now I had been proud of my B&K 2831E DMM, with its maximum resolution of 0.1 µA (aka 100 nA); however, it clearly wasn't up to the task. Since Microchip is in Phoenix and National Instruments (NI) is practically next door to me in Austin, Texas I called a friendly engineer at NI and dragged the kit over to hook it up to their expensive gear (see Figure 5).

Despite what the datasheet and the user guide led me to expect, we couldn't get the board to consume less than 500 nA under any circumstances (after allowing for offsets and noise). Since that is what the chip is supposed to consume with the RTCC running, the clock was the obvious suspect. But how to turn it off?

A call to a Microchip engineer confirmed that the RTCC does indeed stay on unless you turn it off, which you can only do by commenting out InitRTCC() in main.c. Time for some reprogramming.

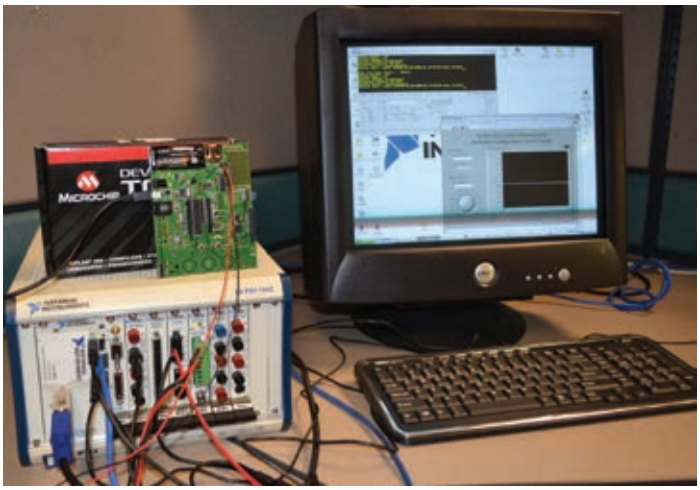


Figure 5: The XLP board gets put to the test.

Programming for nanoamps

To do more with the XLP board than make the lights blink and conduct measurements, you need to reprogram it. You need to download and install Microchip's MPLAB IDE and one or more compilers. I chose version 8.60 of MPLAB IDE (which includes the MPLAB C compiler for PIC24 MCUs and HI-TECHs) instead of the newly announced MPLAB X, which was still in beta at the time. All three installed easily, along with a number of other guides, tools, and libraries. I ordered Microchip's PICkit 3 in-circuit debugger/programmer and a current measurement cable (for JP9), both of which arrived overnight from Digi-Key.

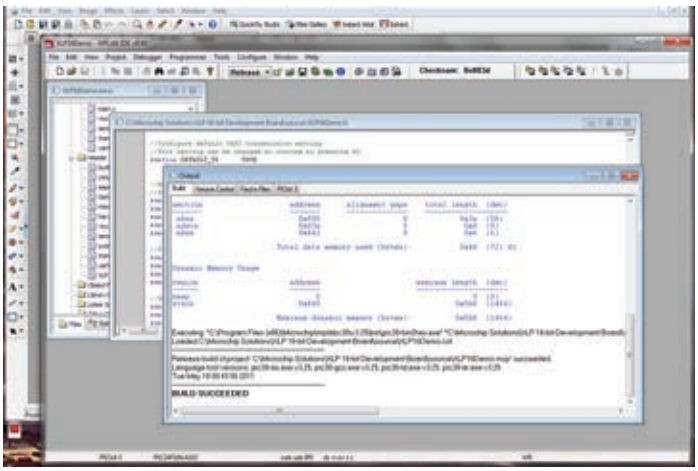


Figure 6: XLP demo program recompiled and ready to run.

Firing up MPLAB, I selected Project/Open, went to directory *…/XLP 16-bit Development Board/source*, and clicked on *XLP16Demo.mcp*. Next, I had to select a compiler from Project/Select Language Toolsuite. I chose the Microchip C30 Toolsuite, which includes the compiler, debugger, linker, assembler, and archiver. There are a lot of other tool options, none of which work until you install them except for the HI-TECH tools, which were also installed along with MPLAB.

Next, I selected Project/Build All (Ctrl-F10) and everything went like clockwork (see Figure 6) – no need to debug just yet. We need to test this program on the board before we start messing with it.

Programmer/Select Programmer/PICkit3 did the job. Microchip warns you not to hot swap the PICkit, so I plugged it into the board and then connected a USB cable to it. MPLAB ran some tests on the programmer, was satisfied, and announced "PICkit3 connected." Well maybe not – moments later I got "PK3E0045: You must connect to a target device to use PICkit3."

I connected another USB cable to the board, which powered it up. MPLAB detected the board but I had to choose the target device. Choosing the PIC24F16KA102 from a long list of supported chips, I got "Target detected" along with a device number. So far, so good. Next "Programmer/Program" quickly led to "Programming/Verify complete." Starting up RealTerm again, the program ran as advertised (see Figure 4). Time to mess with it.

Taking Microchip's advice, I opened main.c, searched out InitRTCC() and commented it out. I rebuilt the program, downloaded it onto the PIC and ran it, watching it perform on the PC. I also hooked up the ammeter on my DMM to the current measurement test point (JP9). Where previously it had at least shown signs of life, now I only got a string of zeros. Time for the acid test using much better test equipment.

The acid test

I revisited my friends at NI, who again hooked the board up to their PXI-4132 Precision Source Measurement Unit (SMU), a $3,500 instrument capable of a 10 pA measurement resolution. I fired up the demo program (with RTCC now disabled) and pressed S2 (MCLR), which promptly shot 300 nA through their device that was calibrated for no more than 100 nA. If it were an analog meter the pointer would have been bent. I promised not to do that again.

After repeated (and forewarned) button pushing, the lowest current we could measure was just under 50 nA, double what the spec sheet claimed (20 nA in Deep Sleep, 25 nA in Sleep). What was that about?

As we pored over the datasheet, NI's guess was that Microchip would have specified the chip at 1.8 V – its lowest V_{DD} – and not the 3.32 V_{DD} that we measured. A quick email to Microchip confirmed our suspicions: "You are correct. The 20 nA number you mentioned before is the 1.8 V spec. At 3.3 V our typical specs are 100 nA for Sleep and around 40 nA for Deep Sleep." Our own measurements were slightly higher but pretty close to the mark.

But wait, there's more

Actually there isn't, at least not in this review. However, the XLP board includes three capacitive sense buttons which you can program for a variety of functions. Its PICtail connector (J7) accepts a number of daughter cards, including ZigBee and MiWi RF boards. It has a connector (J8) that accepts a Cymbet energy scavenging board with a built-in solar panel. Software and firmware support for energy harvesting come with the XLP board, including a full energy harvesting demo with source code. When packaged together it's known as the XLP 16-bit Energy Harvesting Development Kit. We'll reserve that for a future review.

In short, the XLP 16-bit Development Board is an inexpensive ($60), highly extensible platform for developing ultra-low-power applications.

Sources

1. Microchip XLP 16-bit Development Board

Advanced MCUs Anchor Complex Firmware Stack For Sophisticated Field-Oriented Control Motor Designs

by Stephen Evanczuk, *Electronic Products*

Permanent-magnet synchronous motors (PMSM) offer significant advantages in efficiency, responsiveness, and lifecycle cost savings for demanding motor-control applications. Unlike classical brush-type motors, PMSMs are electronically commutated motors (ECM) that require high-integration microcontrollers capable of executing sophisticated Field-Oriented Control (FOC) algorithms and delivering the precise motor-control signals needed to achieve maximum torque.

By drawing on suitable motor-control software libraries, engineers can more easily exploit the advantages of FOC PMSMs using specialized versions of MCUs from leading MCU vendors including Freescale Semiconductor, Microchip Technology, NXP Semiconductors, STMicroelectronics, and Texas Instruments, among others.

In PMSMs, magnetic fields generated in succession in multiple stator windings spin a permanent-magnet rotor. By maintaining a 90º phase between the magnet field generated from each stator winding and the permanent magnet in the rotor, electronic-commutation control logic can achieve maximum torque in the motor. FOC, also known as vector control, provides a means to maintain this optimum phase relationship between stator and rotor by providing high torque even at low speeds and while responding quickly to dynamically changing loads, such as those in washing machines.

FOC approach

FOC continuously commutates the PMSM with sine waves to deliver high torque with little torque ripple. As a result, FOC-driven PMSMs run with reduced mechanical oscillations, outperforming trapezoidal or even sine-wave-driven brushless DC (BLDC) motors – resulting in quieter appliances. Furthermore, FOC-driven PMSMs are highly efficient, thanks to their use of permanent magnets, making them increasingly attractive as energy costs continue to climb – according to the U.S. Department of Energy, kitchen and laundry appliances account for about one-third of household electricity consumption. At the same time, the availability of sensorless methods for ascertaining rotor position further reduces component and lifecycle costs. This opens door to PMSM applications in more demanding compressor and pump designs, such as household, commercial, and automotive applications where sensors cannot easily be used.

FOC implementation imposes significant processing demands, requiring a firmware architecture that combines chip-specific and motor-control-specific software libraries, correspondingly robust processing horsepower, and appropriate integrated peripheral functionality (see Figure 1). Typically, required on-chip peripheral functionality includes multichannel pulse-width modulation signal outputs, with dead time to avoid shoot-through current in the motor-control power stage, high-speed, high-resolution analog-to-digital converters (ADCs) with precision triggering capabilities for accurate measurement of the three-phase current with minimal MCU loading, and fault handling able to shut down the motor quickly in the event of mechanical problems.

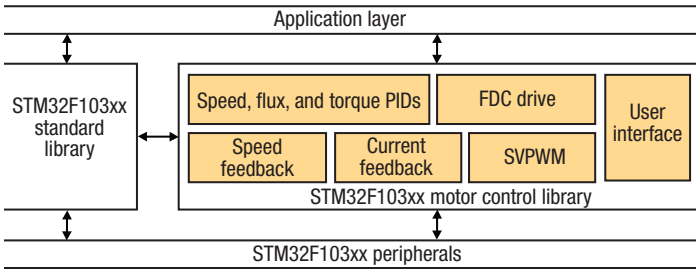


Figure 1: In this representative firmware stack for a Field-Oriented Control architecture, on-chip peripherals complement software routines executed by the device's MCU.

Individual MCU offerings augment these base motor-control capabilities with specialized features designed for broader application requirements and higher performance. For example, Freescale's Cortex-M4 K10 Kinetis MCUs offer dedicated signal-processing features like barrel shifters as well as parallelizing features such as single-cycle, single-instruction/multiple-data (SIMD), and single-cycle multiplier-accumulators (MAC). Microchip's dsPIC33F 16-bit motor control family achieves required performance through its on-chip 40-MIPS digital signal controller core. NXP's LPC17xx and LPC32x0 combine dedicated motor-control PWMs with ARM 32-bit Cortex-M3 and 16/32-bit ARM9 cores, respectively. In each case, MCU suppliers combine specialized, on-chip hardware with specialized software libraries to meet motor-control requirements such as FOC-based designs.

FOC algorithm

The FOC algorithm delivers PMSM advantages by taking three-phase motor signals through a series of transforms designed to translate the three-phase, time-varying motor current measured at the stator into a much simpler two-phase, time-invariant, rotor-centric reference that can be more easily manipulated (see Figure 2).

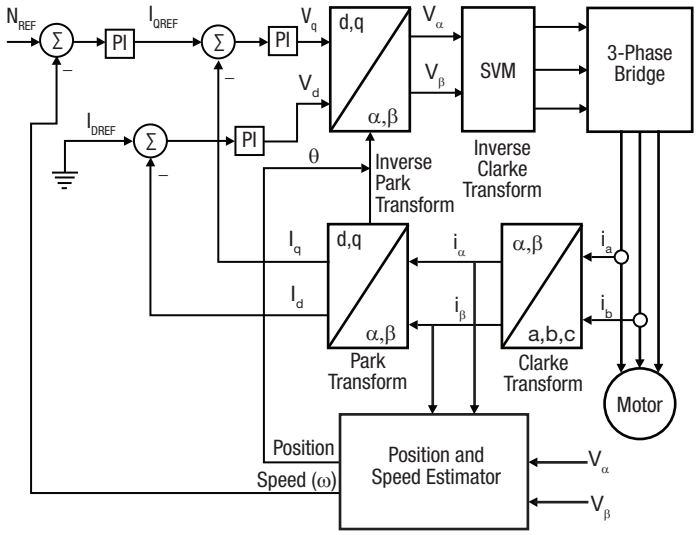


Figure 2: The Field-Oriented Control algorithm uses a series of signal transforms to achieve electronic commutation in permanent magnet synchronous motors.

In the initial input stage of the FOC algorithm, the MCU uses the Clarke transform to project the three-phase (IA, IB, IC) current vector to a two-phase (Iα,Iβ) vector space, which is still referenced to the stator. The MCU then applies the Park transform to rotate the stator-referenced Iα,Iβ vector by angle ϕ to a new two-axes coordinate system with separate flux (I_d) and torque (I_q) components aligned with the rotator. Since this new coordinate system remains fixed, with respect to the rotor, it is time-invariant in that reference plane.

The rotation angle ϕ needed to align the stator field vector to the rotor coordinate system depends, of course, on the rotor position, which has traditionally been measured through sensors attached to the motor shaft. As mentioned above additional sensors increase BOM costs and degrade motor-system life span, motivating increased interest in sensorless approaches which calculate rotor position by leveraging its known relationship with back EMF. Unlike trapezoidal motors, however, back EMF in FOC designs must be estimated using virtual motor models because all phases are engaged in commutation. Consequently, sensorless FOC designs place additional processing load and require MCUs capable of rapidly returning results of trigonometric functions calculated through table lookup, floating-point arithmetic, or integer methods such as the CORDIC algorithm using barrel shifters.

By moving the three-phase, time-varying vector space to a two-axes time-invariant vector space, the FOC algorithm allows engineers to apply conventional proportional-integral-derivative (PID) controllers, used in dc motors, to calculate new target voltages V_d and V_q. In practice, the slow response time of motor speed changes means that D terms are often not needed in these controllers. Note that the use of a permanent magnet in the rotor means that there is no slip because the rotor flux produced by the rotor rotates at the same speed as the rotor field. Consequently, the reference flux I_{DREF} illustrated in Figure 2 is zero; the torque reference Iqref is set through calibration to meet desired performance targets.

The MCU passes the output from the PI controllers through an Inverse Park Transform to translate the rotor-referenced V_d,V_q values to stator-referenced values. The MCU then passes these results through an Inverse Clarke Transform to produce the required three-phase voltage vector, which is then converted to pulse-width modulation (PWM) signals and delivered to a three-phase power stage that directly drives the PMSM.

Motor-control MCU suppliers simplify firmware development for FOC-based designs with extensive software libraries. For example, STMicroelectronics pairs its STM32F103xx family of MCUs with a complete PMSM FOC library written in C. Similarly, TI's TMS320C2 code libraries include optimized C libraries for Park and Clarke transforms able to complete Park transforms, in as little as 125 cycles.

This series of transforms and control functions presents a substantial processing load which is further exacerbated in sensorless designs. Sensorless FOC logic needs to include special initialization and start-up routines to account for lack of observable current measurement in a motor at rest (see Figure 3). FOC startup routines typically apply preset sine-wave patterns that run the motor up to sufficient speed needed for the MCU-based control system to switch to dynamic sensorless measurements.

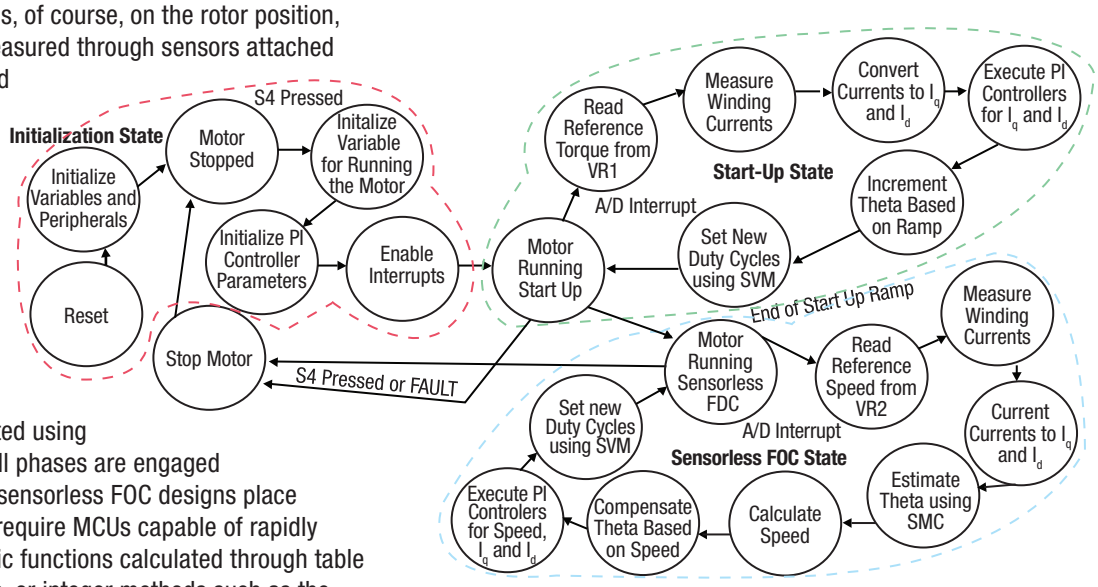


Figure 3: Hitting the start or reset button on a sensorless FOC PMSM-based white-goods application places the motor-control system in specialized states required to properly spin-up the motor to sufficient speeds to permit safe transitions to the sensorless state.

Designing Multithreaded and Multicore Systems

contributed by XMOS

If your MCU application needs to handle digital audio, consider taking a multithreaded approach. Using a multithreaded design approach enables the designer to reuse parts of their design in a straightforward manner.

Multicore and multithreading are efficient methods for designing real-time systems. Using these techniques, a system is designed as a collection of many tasks which operate independently and communicate with each other when required. Breaking the system design down from large monolithic blocks of code into much more manageable tasks greatly simplifies system design and speeds product development. As a result, the real-time properties of the system as a whole are more easily understood. The designer only has to worry about the fidelity of the implementation of each task, asking questions such as, "Is the network protocol implemented correctly?"

In this article, we discuss how to use a multithreaded or multicore design methods to design real-time systems that operate on streams of data, such as digital audio systems. We use several digital audio systems to illustrate the design methods, including asynchronous USB Audio 2, AVB over Ethernet, and digital docks for MP3 players. We briefly discuss the notions of digital audio, multicore, and multithreading before showing how to effectively use multicore and multithreading to design the buffering and clocking schemes required.

Digital audio

Digital audio has taken over from analog audio in many consumer markets for two reasons. First, most audio sources are digital. Whether delivered in lossy compressed form (MP3) or in uncompressed formats (CD), digital standards have taken over from the traditional analog standards such as cassettes and tapes. Second, digital audio is easier to deal with than analog audio. Data can be transferred without loss over existing standards, such as IP or USB, and the hardware design does not need any "magic" to keep the noise floor down. As far as the digital path is concerned, the noise floor is constant and immune from TDMA noise which mobile phones may cause.

A digital audio system operates on streams of samples. Each sample represents the amplitude of one or more audio channels at a point in time, with the time between samples being governed by the sample rate. CD standards have two channels (left and right) and use a sample rate of 44.1 kHz. Common audio standards use 2, 6 (5.1), and 8 (7.1) channels, and sample rates of 44.1 kHz, 48 kHz, or a multiple. We use 48 kHz as a running example, but this is by no means the only standard.

Multicore and multithreading

In a multithreaded design approach, a system is expressed as a collection of concurrent tasks. Using concurrent tasks, rather than a single monolithic program, has several advantages:

- Multiple tasks are a good way to support separation of concerns, which isone of the most important aspects of software engineering. Separation of concerns means that different tasks of the design can be individually designed, implemented, tested, and verified. Once the interaction between the tasks has been specified, teams or individuals can each get on with their own tasks.
- Concurrent tasks provide an easy framework to specify what a system should be doing. For example, a digital audio system will play audio samples that are received over a network interface. In other words, the system should concurrently perform two tasks: receive data from the network interface and play samples on its audio interface. Expressing these two tasks as a single sequential task is confusing.

A system that is expressed as a collection of concurrent tasks can be implemented by a collection of threads in one or more multithreaded cores (see Figure 1). We assume that threads are scheduled at the instruction level, as is the case on an XMOS XCore processor, because that enables concurrent tasks to operate in real-time. Note that this is different from multithreading on Linux, for example, where threads are scheduled on a uniprocessor with context switching. This may make those threads appear concurrent to a human being, but not to a collection of real-time devices.

Concurrent tasks are logically designed to communicate by message passing, and when two tasks are implemented by two threads, they communicate by sending data and control over channels. Inside a core, channel communication is performed by the core itself and when threads are located on separate cores, channel communication is performed through switches (see Figure 2).

Multithreaded design has been used by embedded system designers for decades. To implement an embedded system, a system designer used to employ a multitude of microcontrollers. For example, inside a music player one may find three microcontrollers controlling the flash, the DAC, and an MP3 decoder chip.

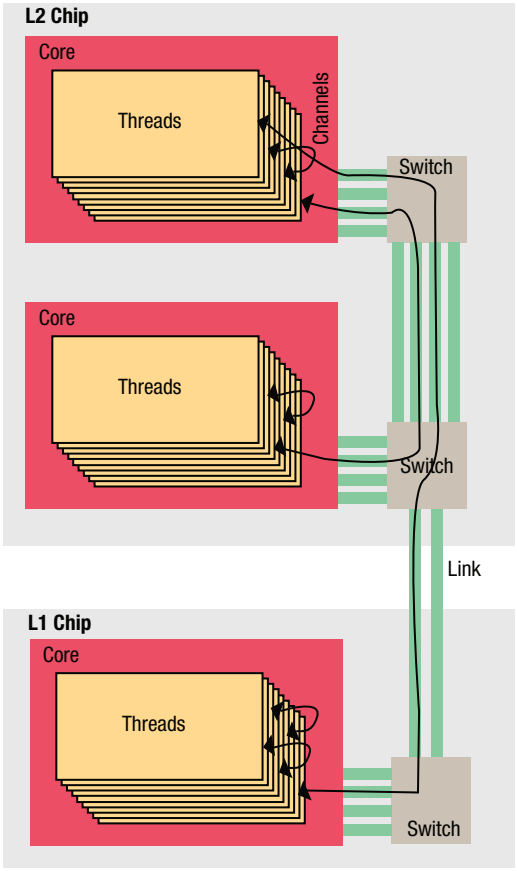


Figure 1: Threads, channels, cores, switches, and links. Concurrent threads communicate over channels either inside a core, between cores on a chip, or between cores on different chips.

We argue that modern day multithreaded environments offer a replacement for this design strategy. A single, multithreaded chip can replace a number of MCUs and provide an integrated communication model between tasks. Instead of having to implement bespoke communication between tasks on separate MCUs, the system is implemented as a set of threads which communicate over channels.

Using a multithreaded design approach enables the designer to reuse parts of their design in a straightforward manner. In traditional software engineering, functions and modules are combined to perform complex tasks. However, this method does not necessarily work in a real-time environment because executing two functions in sequence may break the real-time requirement of either the function or module.

In an ideal multithreaded environment, the composition of real-time tasks is trivial, as it is just a case of adding a thread (or a core) for every new real-time task. In reality, the designer will have constraints on the number of cores (for example, for financial reasons) and hence will have to make a decision about which tasks to compose as concurrent threads, and which tasks to integrate in a single thread as a collection of functions.

Multithreaded digital audio

A digital audio system is easily split into multiple threads, including, a network protocol stack thread, a clock recovery thread, an audio delivery thread, and optionally, threads for DSP, device upgrade, and driver authentication. The network protocol stack may be as complex as an Ethernet/IP stack and comprise multiple concurrent tasks, or as simple as an S/PDIF receiver.

Figure 2: Physical incarnation of a three-core system with 24 concurrent threads. The top device has two cores and the bottom device has a single core.

We assume that the threads in the system communicate by sending data samples over channels. Whether the threads execute on a single core or on a multicore system is not important in this design method, since multicore just adds scalability to the design. We assume that the computational requirements for each thread can be established statically and are not data dependent, which is normally the case for uncompressed audio.

We will focus our attention on two parts of the design: buffering between threads (and their impact on performance) and clock recovery. Once these design decisions have been made, implementing the inside of each thread follows normal software engineering principles, and is as hard or easy as one would expect. Buffering and clock recovery are interesting because they both have a qualitative impact on the user experience (facilitating stable low latency audio) and they are easily understood in a multithreaded programming environment.

Buffering

Within a digital solution, data samples are not necessarily transported at the time that they are to be delivered. This requires digital audio to be buffered. As an example, consider a USB 2.0 speaker with a 48 kHz sample rate. The USB layer will transport a burst of six samples in every 125 µs window. There is no guarantee where in a

125 µs window the six samples will be delivered, hence a buffer of at least 12 samples is required in order to guarantee that samples can be streamed out in real-time to the speaker.

The design challenge is to establish the right amount of buffering. In an analog system, buffering is not an issue; the signal is delivered on time. In a digital system designed on top of a non-real-time OS, programmers usually stick to a reasonably large buffer (250 or 1,000 samples) in order to cope with uncertainties in scheduling policies. However, large buffers are costly in terms of memory, in terms of adding latency, and in terms of proving that they are large enough to guarantee click-free delivery.

Multithreaded design provides a good framework to informally and formally reason about buffering and avoids unnecessarily large buffers. As an example, consider the above USB speaker augmented with an ambient noise correction system. This system will comprise the following threads:

- A thread that receives USB samples over the network.
- A series of 10 or more threads that filter the stream of samples, each with a different set of coefficients.
- A thread that delivers a filtered output sample to the stereo codec using I^2S.
- A thread that reads samples from a codec connected to a microphone sampling ambient noise.
- A thread that subsamples the ambient noise to an 8 kHz sample rate.
- A thread that establishes the spectral characteristics of the ambient noise.
- A thread that changes the filter coefficients based on the computed spectral characteristics.

All threads will operate on some multiple of the 48 kHz base period. For example, each of the filtering threads will filter exactly one sample every 48 kHz period; the delivery thread will deliver a sample every period. Each of the threads also has a defined window over which it operates, and a defined method by which this window is advanced. For example, if our filter thread is implemented using a biquad, it will operate on a window of three samples which is advanced by one sample every period. The spectral thread may operate on a 256 sample window (to perform an FFT (Fest Fourier Transform)) which is advanced by 64 samples every 64 samples.

One can now establish all parts of the system that operate on the same period and connect them together in synchronous parts. No buffers are required inside those synchronous parts, although if threads are to operate in a pipeline, single buffers are required. Between the various synchronous parts buffers are required. In our example, we end up with three parts:

1. The part that receives samples from USB, filters, and delivers at 48 kHz.
2. The part that samples ambient noise at 48 kHz and delivers at 8 kHz.
3. The part that establishes the spectral characteristics and changes the filter settings at 125 Hz.

These three parts are shown in Figure 3. The first part that receives samples from the USB buffer needs to buffer 12 stereo samples.

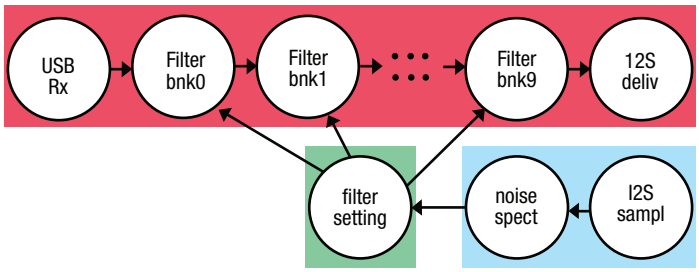


Figure 3: The threads grouped together based on their frequency.

The part that delivers needs to buffer one stereo sample. Operating the 10 filter threads as a pipeline requires 11 buffers. That means that the total delay from receiver to codec comprises 24 sample times, or 500 µs, and an extra sample can be added to cope with medium term jitter in the clock recovery algorithm. This part runs at 48 kHz.

The second part that samples ambient noise needs to store one sample on the input side and six samples for subsampling. Hence, there is a seven sample delay at 48 kHz, or 145 µs.

The third part that establishes the spectral characteristics needs to store 256 samples, at an 8 kHz sample rate. No other buffers are required. Hence, the delay between ambient noise and filter correction is 256 samples at 8 kHz and 145 µs for the subsampling, or just over 32 ms. Note that these are minimum buffer sizes for the algorithm that we have chosen to use; if this latency is unacceptable, a different algorithm has to be chosen.

There is often a temptation to design the threads to operate on blocks of data instead of single samples, but this will increase the overall latency experienced, increase the memory requirements, and increase complexity. This should only be considered if there is a clear benefit, such as an increased throughput.

Clocking digital audio

A big difference between digital and analog audio is that analog audio is based on this underlying sample rate and digital audio requires a clock signal to be distributed to all parts of the system. Although components can all use different sample rates (for example, some parts of the system may use 48 kHz and some other parts may use 96 kHz with a sample rate converter in between), all components should agree on the length of a second, and therefore agree on a basis to measure frequencies.

An interesting property of digital audio is that all threads inside the system are agnostic to the base of this clock frequency, assuming that there is a gold standard base rate. It does not matter if multiple cores in the system use different crystals, as long as they operate on a sample. However, at the edges of the system, the true clock frequency is important and the delay that a sample has incurred en route becomes important.

In a multithreaded environment, one will set a thread aside to explicitly measure the true clock frequency, implement the clock recovery algorithm, measure the local clock versus the global clock, and agree with a master clock on the clock offset.

The clock may be measured implicitly using the underlying bit rate of interconnects, such as S/PDIF or ADAT. Measuring the number of bits per second on either of those networks will give a measure for the master clock. The clock may be measured explicitly by using protocols designed for this purpose, such as PTP over Ethernet.

In the clock recovery thread, a control loop can be implemented, which estimates the clock frequency, and adjusts based on the error observed. In its simplest form, the error is used as a metric to adjust the frequency, but filters can be used to reduce jitter. This software thread implements what would have been traditionally performed by a PLL but in software, and hence it can be adjusted to the environment cheaply.

Conclusion

A multithreaded development method enables digital audio systems to be developed using a divide and conquer approach, where a problem is split into a set of concurrent tasks that are each executed in a separate thread on a multithreaded core.

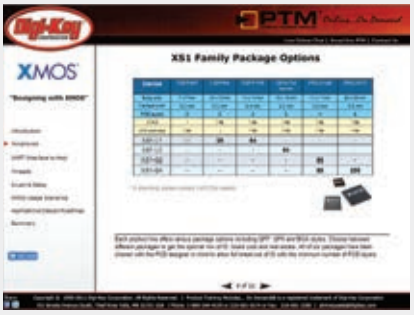
Like many real-time systems, digital audio lends itself to a multithreaded design method because digital audio systems obviously consist of a group of tasks that work on data and also require those tasks to execute concurrently.

Designing with XMOS

XMOS

The heart of the XMOS is the Xcore. It is a proprietary 400 MHz, 32-bit processor core that supports single-cycle instruction execution, hardware multi-threading, intelligent ports that are tightly coupled, and more.

The Designing with XMOS PTM features an overview of the architecture, tools, and applications of the XMOS. The module covers XS1 technology, provides insight into design tools, gives extensive multithreading examples, and lists applications for XS1.



XMOS®

PTM PRODUCT TRAINING MODULES *Online...On Demand®*

www.digikey.com/ptm

BOLDLY GOING WHERE NO MICROCONTROLLER HAS GONE BEFORE.

FRAM FROM TI.
CHANGING ULTRA-LOW POWER MEMORY FOREVER

TEXAS INSTRUMENTS

- Leading ultra-low-power operation – as low as 100 µA/MHz
- Unlimited write endurance
- Write guarantee
- "No Power" write
- Fastest memory access
- Superior security
- Radiation resistant
- Enhanced reliability
- Backwards code compatible with ALL MSP430™ devices

Brought to you by MSP Microcontrollers from TI – the world's lowest power microcontroller.

www.digikey.com/ti-mcu

The Z8 Encore!® MCU as an LCD Driver

contributed by Zilog

Driving a multiplexed, 4-segment LCD is a straightforward matter using an 8-bit MCU, a small BOM, and the downloadable code provided in this article.

Directly driving an LCD can be accomplished without a dedicated LCD driver on the microcontroller, and requires few additional resources. There are two types of LCDs – static and multiplexed. This article discusses the programming of the multiplexed LCD in conjunction with Zilog's Z8 Encore!® microcontroller.

Z8 Encore! flash microcontrollers overview

Zilog's Z8 Encore! products are based on the eZ8™ CPU and introduce flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance, register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8 MCU.

The Z8 Encore! MCUs combine a 20 MHz core with flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! MCU suitable for various applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Driving LCDs

A static LCD features a separate pin for each segment of the LCD and a common backplane pin. A requirement for illuminating a segment is to bias the segment in opposition to the backplane. An additional requirement is that LCDs cannot allow direct current (DC) to be present on a segment.

To prevent DC on a segment, the backplane is driven with a low-frequency square wave, and the segments are toggled with respect to the backplane.

Multiplex LCDs feature multiple backplanes, and a single segment pin is shared among multiple segments. To illuminate a particular segment, the segment pin is driven in opposition to the backplane, and the unused backplanes remain in an IDLE state. The backplanes are again driven with a low-frequency square wave to prevent DC bias on the segments.

The challenge

Programming a multiplexed LCD can be a difficult task due to the multiplexed arrangement of the segments. Multiplexed LEDs usually feature a separate backplane for each LED digit. However, multiplexed LCDs arrange their backplanes across the top, middle, and bottom of the digit. This arrangement can make the decoding process very complicated, but it is important to mention that even microcontrollers that feature dedicated LCD drivers still require a difficult decoding process (see Figure 1).

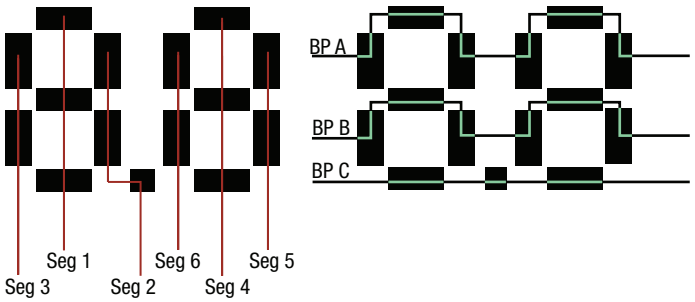


Figure 1: Segment arrangement.

The next engineering task is related to the voltage required to illuminate a segment. The ON drive level of a multiplexed segment is reduced because the segment spends most of its time in an IDLE state and is only asserted 25 percent of the time. In effect, this statement means that at lower operating voltages, a segment may not illuminate.

A further complication is that the segment can perceive a voltage potential while in its OFF state as the shared segment pin is being asserted by the currently active backplane. These contrasting problems may become worse as more backplanes are added to the display, because with each additional backplane, the available ON voltage is reduced, and the residual OFF voltage is increased.

Figure 2 displays how contrast decreases with each backplane, due to the fact that there is less difference between an ON segment and an OFF segment. In Figure 2, a static display with one backplane receives 100 percent of its available V_{cc} for an ON voltage and 0 percent for an OFF voltage. In the three-plane example, one-half of the V_{cc} is available for ON voltage and an OFF segment receives one-quarter of the V_{cc}. LCDs vary from one manufacturer to the next, but the typical threshold voltage is 2.3 V RMS.

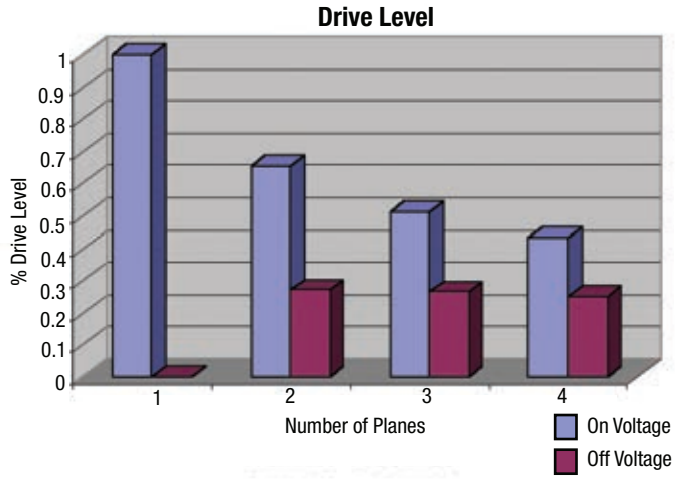


Figure 2: Drive level.

With only one-half of the V_{cc} available for ON voltage, it is easy to see how a 3.3 V microcontroller is unable to directly drive a multiplex LCD. The purpose of this article is to show that you can drive a multiplexed LCD on the 3 V Z8 Encore! MCU.

Hardware architecture

To drive a multiplexed LCD with a 3 V MCU, the drive level must be boosted. To reduce gate count and complexity, only the backplanes are boosted. Segment drive voltages swing above and below one-half V_{cc}; therefore, a boosted backplane signal must perform in the same manner by using the Z8 Encore! MCU's port pins as two charge pumps referenced at one-half V_{cc}. IC1, the 4050 buffer is used to provide the level-shifting function.

Each backplane is driven high, and idled while the other planes are driven. The process is inverted to remove any DC component, as shown in Figure 3.

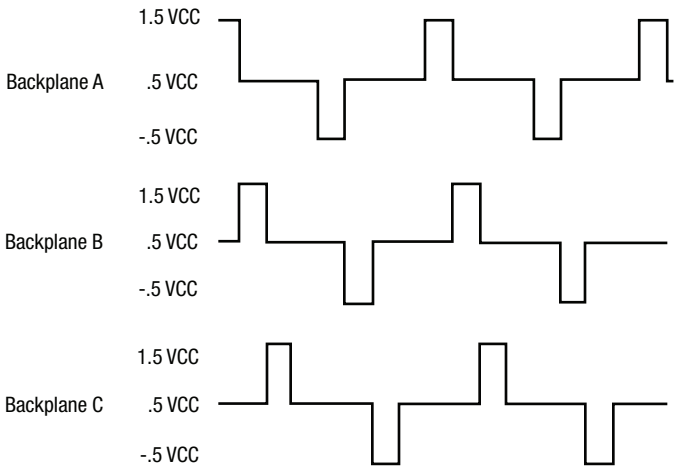


Figure 3: Backplane waveforms.

Segments are turned ON by driving the segment pin in the opposite direction of the active backplane, and OFF by driving the pin in the direction of the active backplane. During the backplane's IDLE state, the voltage on any segment is below the threshold voltage. As a result, the segments remain unlit.

Software implementation

Due to the additional speed and memory of the Z8 Encore! family, it is assumed that development occurs in the C programming language. Applications written in C can be easily ported to different environments, if the software is written to allow easy porting.

With this in mind, macros are used for the I/O-specific operations so that the bulk of the software will remain untouched if the code is ported to another device.

The following code segment maintains the charge pumps.

```
/*Charge pump definitions
The charge pumps boost the segment drive
voltage and are serviced each timer
interrupt. The Positive pump is pulled Low
to charge and floated to an input state.
The negative pump is floated when charging.
The cap is referenced at 1/2 VCC and the
charge on the capacitor appears to be
VCC +/- the reference. The macro also
initializes the port mode for the port so
it is always refreshed.
*/

#define ChargePumpsPDADDR=PxHD;
PDCTL|= B3|B4; PDOD&=~B3;
PDOD|=B4; PDADDR=PxOC;
PDCTL&=~(B3|B4); PDADDR=PxDDR;
PDCTL&=~(B3|B4)
#define FloatPumpsPDADDR=PxDDR; PDCTL|=B4|B3
```

The following macros manage the backplane drive. These macros are complex, as only one pin is required per backplane, but two pin states are required for each backplane.

```
/*Backplane drives require three states: an
ON, an OFF, and an IDLE. By mixing BP1 with
BP2, BP2 with BP3, and BP3 with BP1 it's
possible to get all three states on each
plane without requiring additional pins.

PlaneX123

_______________
BP11101
BP20110
BP31011
`BP10010
`BP21001
`BP30100

*/

#define SetUpBackplanePDADDR=PxOC;
PDCTL&=~(B0|B5|B6);
PDADDR=PxDDR; PDCTL&=~(B0|B5|B6)
#define BP1PDOD&=~B6; PDOD|=B0|B5
#define BP2PDOD&=~B5; PDOD|=B0|B6
#define BP3PDOD&=~B0; PDOD|=B6|B5
#define NotBP1PDOD&=~(B0|B5); PDOD|=B6
#define NotBP2PDOD&=~(B0|B6); PDOD|=B5
#define NotBP3PDOD&=~(B5|B6); PDOD|=B0
```

Finally, the macros for driving the segments.

```
/*This next macro takes the individual
segments stored in the display buffer and
places them on the ports. It could have
been done without the macro but this makes
it more generic. There will be six planes
with two buffers because
there are more than 8 segments*/

#define DisplaySegmentsPAOD&=~0xF8;
PAOD|=(buffer[plane]&0x00F8); PCOD=0;
PCOD|=((buffer[plane]&0x7F00)>>8)
```

As mentioned earlier, the difficult programming involved in multiplex LCDs is a rather unusual multiplexing scheme. The previous macro simply places the previously decoded segments from the buffer onto the ports. As the decoding process is so involved, it is not performed in the Interrupt Service Routine (ISR). ISRs must be kept as short as possible; all that is required in the ISR is to set the backplanes and drive the segments. The buffer is an integer array which holds the 12 segments used in our display. The single dimension in this array is the plane. There are six planes in this dimension, one for each backplane state: A, B, C, A', B', and C'. When the decoding process is complete, the buffer must be loaded with the 12 segments of data across the six planes.

The first step in decoding is to define how the characters are displayed. This definition is universal for all seven segment displays. Therefore, the following code segment can be reused.

```
#define Dig_0Seg_a | Seg_b | Seg_c | Seg_d |
Seg_e | Seg_f
#define Dig_1Seg_b | Seg_c
#define Dig_2Seg_a | Seg_b | Seg_g | Seg_e |
Seg_d
#define Dig_3Seg_a | Seg_b | Seg_g | Seg_c |
Seg_d
#define Dig_4Seg_f | Seg_g | Seg_b | Seg_c
#define Dig_5Seg_a | Seg_f | Seg_g | Seg_c |
Seg_d
#define Dig_6Seg_a | Seg_f | Seg_g | Seg_c |
Seg_d | Seg_e
#define Dig_7Seg_a | Dig_1
#define Dig_8Seg_g | Dig_0
#define Dig_9Seg_a | Seg_f | Seg_g | Seg_b |
Seg_c
#define Dig_ASeg_a | Seg_b | Seg_c | Seg_g |
Seg_e | Seg_f
#define Dig_bSeg_f | Seg_e | Seg_g | Seg_d |
Seg_c
#define Dig_CSeg_a | Seg_f | Seg_e | Seg_d
#define Dig_dSeg_b | Seg_c | Seg_d | Seg_e |
Seg_g
#define Dig_ESeg_a | Seg_f | Seg_e | Seg_d |
Seg_g
#define Dig_FSeg_a | Seg_f | Seg_e | Seg_g
#define Dig_gSeg_a | Seg_f | Seg_g | Seg_b |
Seg_c | Seg_d
#define Dig_hSeg_g | Seg_c | Seg_e | Seg_f
#define Dig_IDig_1
#define Dig_JDig_1 | Seg_d
#define Dig_LSeg_d | Seg_e | Seg_f
#define Dig_nSeg_c | Seg_e | Seg_g
#define Dig_ODig_0
#define Dig_PSeg_g | Seg_a | Seg_b | Seg_e |
Seg_f
#define Dig_rSeg_g | Seg_e | Seg_f
#define Dig_SSeg_g | Seg_a | Seg_c | Seg_d |
Seg_f
#define Dig_tSeg_g | Seg_e | Seg_f | Seg_d
#define Dig_USeg_b | Seg_c | Seg_d | Seg_e |
Seg_f
```

The segments are scattered across three separate backplanes, and must be arranged in a single byte as three packets of three bits. The last bit is unused. This arrangement mirrors the physical layout of a display digit (see Table 1).

Table 1: Segment assignment.

Backplane	A	B	C
Segment 1	a	g	d
Segment 2	b	c	decimal
Segment 3	f	e	Not Used

```
#define Seg_a1
#define Seg_g2
#define Seg_d4

#define Seg_b8
#define Seg_c16
#define Seg_dp32

#define Seg_f64
#define Seg_e128
```

Our goal is to place the segment data into three integers – one for each display backplane. The software must split the single byte representing the seven-segment character into the three segments by three planes. Table 2 indicates how the integer array stores the individual segment bits.

Table 2: Segment array.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PlaneA					4f	4b	4a	3f	3b	3a	2f	2b	2a	1f	1b	1a
PlaneB					4e	4c	4g	3e	3c	3g	2e	2c	2g	1e	1c	1g
PlaneC					NC	4dp	4d	NC	3dp	3d	NC	2dp	2d	NC	1dp	1d

As each digit of the LCD requires three segments, addressing the correct segment pin requires shifting the data over three bits for each digit. Finally, the correct physical pin of the microcontroller must be addressed. The assignments in the following code segment can change based upon the board layout and other resources.

```
#define Seg_1AGD8//PAOD|=B3
#define Seg_1BCDP16//PAOD|=B4
#define Seg_1FE32//PAOD|=B5
#define Seg_2AGD64//PAOD|=B6
#define Seg_2BCDP128//PAOD|=B7

#define Seg_2FE256//PCOD|=B0

#define Seg_3AGD512//PCOD|=B1
#define Seg_3BCDP1024//PCOD|=B2
#define Seg_3FE2048//PCOD|=B3

#define Seg_4AGD4096//PCOD|=B4
#define Seg_4BCDP8192//PCOD|=B5
#define Seg_4FE16384//PCOD|=B6
```

The following code segment determines which segments turn ON for a particular character.

```
for (digit=0, shift=9; digit<4; digit++,
shift-=3)
{
      segments[0]|=(0x07 &
CharTbl[que[digit]])<<shift;
      segments[1]|=((0x38 & CharTbl[que[dig
it]])>>3)<<shift;
      segments[2]|=((0x1C0 & CharTbl[que[di
git]])>>6)<<shift;
}
```

Storing the correct segment to turn ON requires testing each individual bit. The advantage is that the code becomes very portable, as shown here:

```
for(plane=0;plane<3;plane++)
{
if (segments[plane]&B0)
      buffer[plane]|=Seg_1AGD;

if (segments[plane]&B1)
      buffer[plane]|=Seg_1BCDP;

if (segments[plane]&B2)
      buffer[plane]|=Seg_1FE;

if (segments[plane]&B3)
      buffer[plane]|=Seg_2AGD;

etc.
}
```

```
/*We have an assignment here rather
than setting a single variable in the
statements above because a timer IRQ
occurs while the segments are being
gathered and that will cause the display
to flicker. The last three buffers are
simply complements of the first three.
*/

buffer[0]=tempbuffer[0];
buffer[1]=tempbuffer[1];
buffer[2]=tempbuffer[2];
buffer[3]=~buffer[0];
buffer[4]=~buffer[1];
buffer[5]=~buffer[2];
```

Summary

The actual code required for directly driving an LCD is not very complex. The time required to decode the individual segment planes in this C example is only 141 µs. The advantage is the ability to drive very large displays directly without an additional LCD driver, or the use of a microcontroller with a dedicated driver. The only disadvantage is the additional pins required for the charge pumps and backplane drive, but in most cases the additional pins are cheaper than a dedicated driver.

Figure 4 shows the schematic diagram for an LCD drive using the Z8 Encore! MCU. Code for this project is available on Zilog's web site at www.zilog.com/docs/z8encore/appnotes/an0162-sc01.zip.

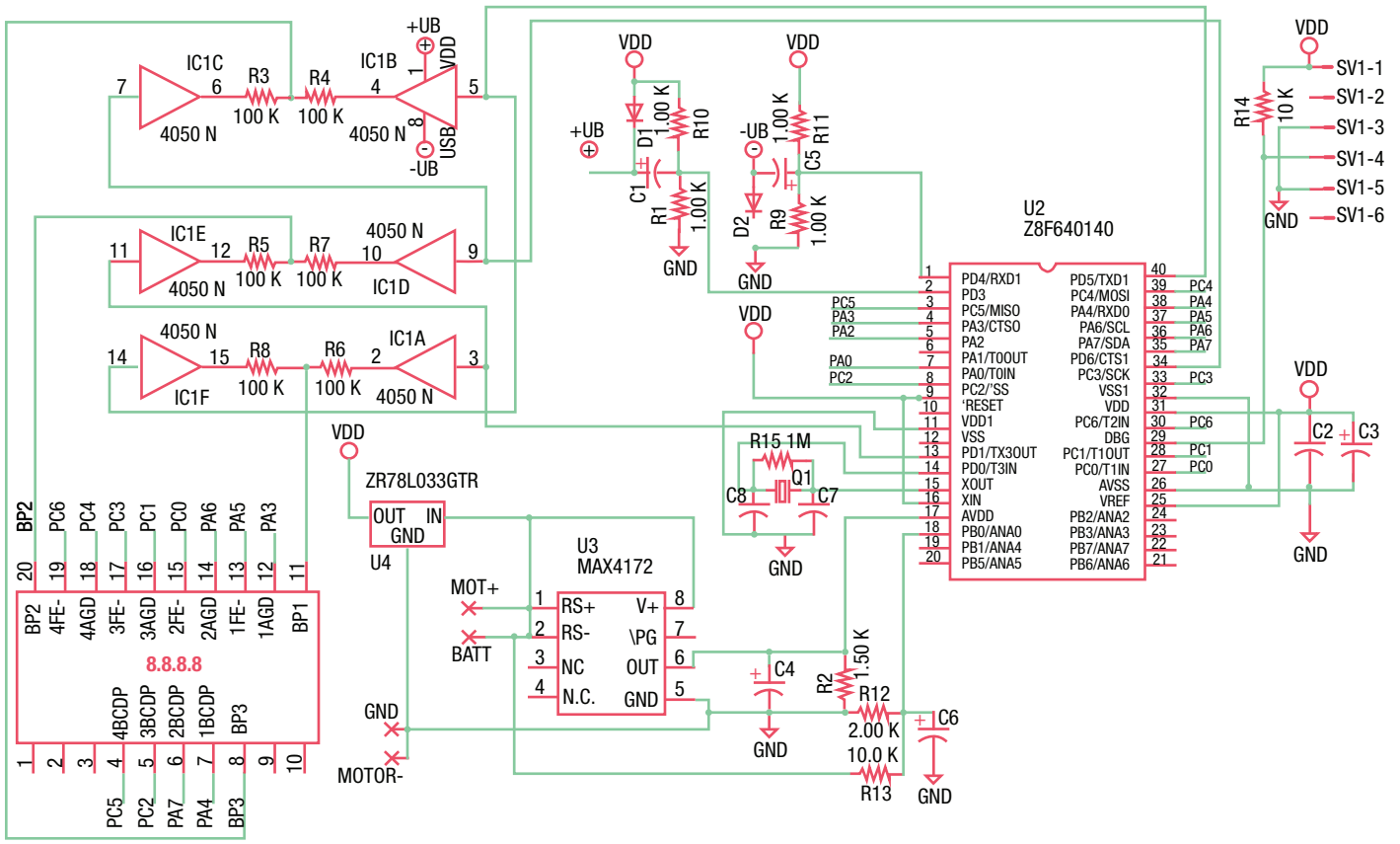
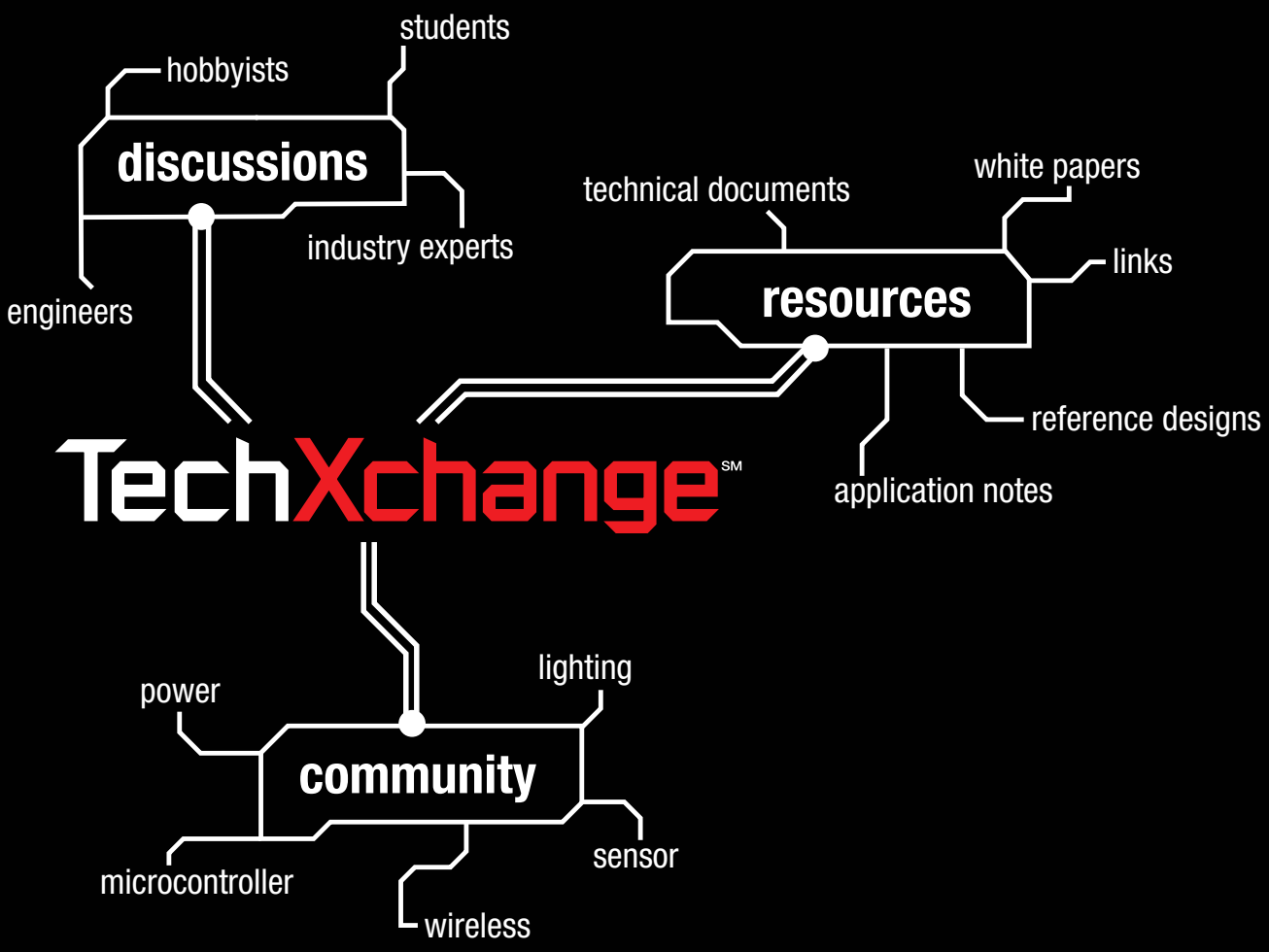


Figure 4: Schematic for an LCD drive using the Z8 Encore! MCU.

It's all about **connections.**



The user-to-user forum is for everyone, from design engineers to hobbyists, to discuss technology, products, designs and more. Join the discussions that match your interest or offer your expertise to others.

Join the discussion now at:

www.digikey.com/techxchange

Digi-Key is an authorized distributor for all supplier partners. New products added daily.

© 2011 Digi-Key Corporation, 701 Brooks Ave. South, Thief River Falls, MN 56701, USA

Supplier Authorized Distributors

Digi-Key® CORPORATION

The industry's broadest product selection available for immediate delivery

WANT MORE?

SUBSCRIBE TODAY!

Get the latest issue of **TechZone™ Magazine** when you want it. Choose from our Lighting, Microcontroller, Sensors, and Wireless Solutions **TechZone Magazines** to remain up-to-date on the latest technologies.

www.digikey.com/request

©2011 Digi-Key Corporation, 701 Brooks Ave. South, Thief River Falls, MN 56701, USA