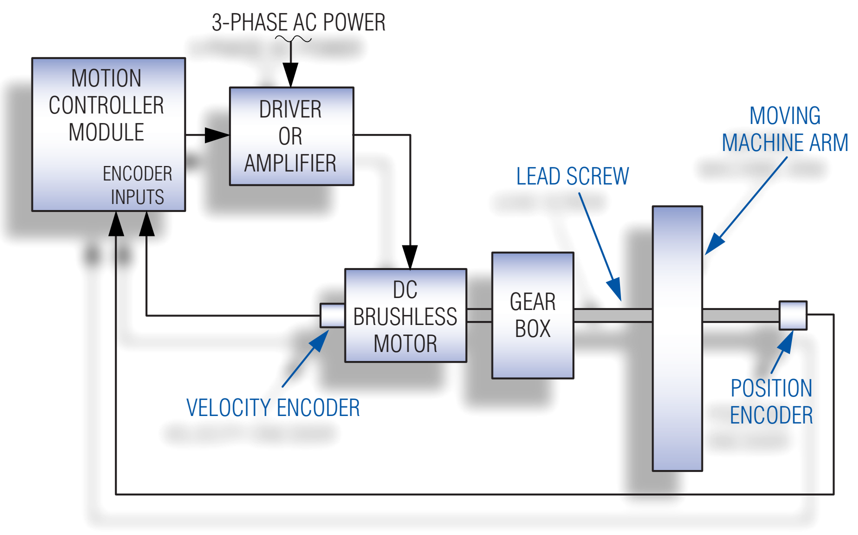
Engineering Journal

DALLAS SEMICONDUCTOR MAXIM

Volume Forty-Eight



This DC brushless servo system uses quadrature optical encoders to provide velocity and position feedback to the motion controller. See article inside, page 3.

MAXIM REPORTS REVENUES AND EARNINGS FOR THE THIRD QUARTER OF FISCAL 2003 AND DOUBLES QUARTERLY DIVIDEND

Maxim Integrated Products, Inc., (MXIM) reported net revenues of $286.2 million for its fiscal third quarter ending March 29, 2003, a 10.7% increase over the $258.5 million reported for the third quarter of fiscal 2002 and unchanged from reported revenues for the second quarter of fiscal 2003. Net income for the quarter was $77.6 million, a 16.3% increase over the $66.7 million reported last year and a 0.7% increase over the $77.1 million reported for the previous quarter. Diluted earnings per share (which measure the cost of employee stock options and include the Company's average outstanding common stock for the quarter, or 322.9 million shares, increased by 19.0 million shares using the Treasury Stock Method) were $0.23 for the third quarter, a 21.1% increase over the $0.19 reported for the same period a year ago and unchanged from reported second quarter fiscal 2003 results.

During the quarter, cash and short-term investments increased $111.0 million after the Company repurchased 650,000 shares of its common stock for $20.0 million, paid dividends of $6.5 million, and acquired a total of $8.6 million of capital equipment. Inventories decreased $4.2 million during the quarter to $123.8 million.

Third quarter bookings were approximately $308 million, a 14% increase over the second quarter's level of $271 million. Turns orders received during the third quarter were $165 million, a 19% increase over the $139 million received in the prior quarter (turns orders are customer orders that are for delivery within the same quarter and may result in revenue within the same quarter if the Company has available inventory that matches those orders). This is the highest level of turns orders that the Company has received since the first quarter of its 2001 fiscal year. Order cancellations were $4.6 million, a decrease of 41% from the prior quarter. Bookings increased over the second quarter's level in all major geographic regions except Japan, where bookings were down 5% sequentially. Twelve of the Company's 14 business units had improved bookings in the third quarter. Eleven of those 14 business units had a bookings increase of at least 10% over the previous quarter.

Third quarter ending backlog shippable within the next 12 months was approximately $219 million, including $196 million requested for shipment in the fourth quarter of fiscal 2003. Second quarter ending backlog shippable within the next 12 months was approximately $201 million, including $177 million requested for shipment in the third quarter of fiscal 2003.

Jack Gifford, Chairman, President, and Chief Executive Officer, commented on the quarter: "We were encouraged to see the increase in orders this quarter, particularly the improvement in bookings by Dallas Semiconductor. Orders improved not only for our power management and communications products, but also for our standard products that have very broad-based markets and applications. The continued increase in turns orders in the third quarter may indicate that our customers are getting increased demand for their products quarter over quarter. We see no signs of inventory build-up either at our distributors or at our end customers and believe that the current shipping level is close to or slightly below the current consumption level for our products."

Mr. Gifford continued: "Based on the Company's profitability, strong cash position, and business outlook, the Company's Board of Directors has increased this quarter's dividend from $0.02 per share to $0.04 per share. Payment will be made on May 30, 2003 to stockholders of record on May 12, 2003."

Mr. Gifford concluded: "Maxim is opposed to expensing employee stock options, as we believe that this concept is bad accounting. We believe that the Treasury Stock Method both rigorously and accurately defines the cost and resulting dilutive effect of stock options on reported earnings. We fear that requiring U.S. corporations to expense stock options could seriously damage our country's leadership position in technology innovation and our ability to effectively compete with other countries. Our nation's technology leadership over the past 15 years has caused some to forget the economic effect during the 1970s of our country's losing our technology leadership to Japan. Entrepreneurial individuals and companies, given incentive by stock options, reversed this situation. This impact on the U.S. economy and its people in the 1970s has been too easily forgotten. The potential damaging long-term effect on our nation's economy and lifestyle resulting from the forced expensing of stock options should be taken very seriously."

Designing robust and fault-tolerant motion-control feedback systems

Today's demanding industrial applications require rugged, reliable robots and automated machines that operate under harsh conditions 24 hours a day, seven days a week. Such systems require far more precise motors and feedback controls than were formerly necessary, and much of the improved performance available today can be attributed to newer technologies and to microelectronics. Those innovations provide more robust automated systems by eliminating robot collisions in shared work-spaces, improving task assignments, and honing servo accuracies.

The key to robust operation in a system lies in the way it handles mechanical and electrical faults. This article discusses the design of a robust and fault-tolerant motion-control system whose feedback paths incorporate quadrature encoders.

Servo Systems

Modern automated systems incorporate closed-loop feedback for motion control. They typically include a servo system that consists of a motor driver and feedback elements combined in a manner that gives accurate and stable control over speed and position. The various system-level components of a typical servo system are illustrated in **Figure 1**.

DC brushless motors are preferred for high-performance and high-speed applications. DC brush and stepper motors are suitable for low-speed and less-demanding applications. Brushless motors are assumed throughout this article. Such motors typically include a quadrature encoder on the end shaft that determines the shaft velocity and commutation point for controlling the motor's coil-switching sequences (see sidebar, *Feedback encoder types*). A second quadrature encoder on the machine's rotating shaft provides position data for that shaft, which generally differs from the motor-shaft position due to inaccuracies caused by backlash in the gearhead and lead-screw assemblies.

Typical motion-controller cards and modules include a motion-control IC, a microprocessor, and a DSP or custom ASIC for processing the high-speed encoder signals. The controller provides velocity and direction-of-rotation signals to the driver or amplifier, which in turn provides the proper levels of voltage and current (power) to operate the motor. To design a robust and fault-tolerant motion-control system with feedback, you must address the following items at the system level during design:

- Controller-encoder input circuits (receiver circuits)
- Receiver circuit PC board layout
- Encoder-signal cabling system

Though not addressed in this article, motion-controller inputs, such as hard-wired emergency stop and limit inputs, should also be considered when designing a fault-tolerant feedback system.

Controller receiver circuits

The motor's quadrature encoder sends six RS-422/RS-485 signals (A, A\, B, B\, Index, and Index\) down the cable to the motion controller's receiver circuit (the encoder input). The receiver converts the RS-422 signals to logic-level signals (we assume RS-422 signals because the system has only one transmitter), and feeds them to the motion-controller circuit for processing. (For RS-422 and RS-485 differences, refer to the short online tutorial *RS-485 Basics* at www.maxim-ic.com.) The receiver circuit must respond to various faults in the servo-system environment, including opens, shorts, and noise. See the sidebar *Fault types* for information on faults and ESD.

Figure 2 illustrates the encoder-input receiver circuit in a typical motion controller. U1 is a 10Mbps, 5V, quad RS-422/RS-485 receiver with ±15kV ESD protection. For fault-tolerant systems with encoder inputs that connect

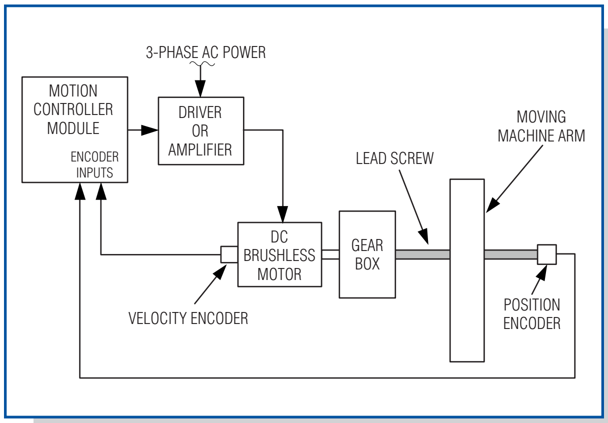


Figure 1. This DC brushless servo system uses quadrature optical encoders to provide velocity and position feedback to the motion controller.

to external components, ESD protection is a must. The absence of external ESD-protection components substantially reduces the PC board area required for this circuit.

The 150Ω resistors provide proper terminations for each group of complementary signal pairs transmitted down the twisted-pair cable from the quadrature encoder. (Cable termination and related issues are covered in detail later.) A break or disconnect in the cable produces an open-circuit fault that must be detected before the motion controller can take appropriate action. As a fail-safe measure, the MAX3095 receiver outputs assert a logic high in response to an open-circuited input pair. The 1kΩ resistors bias the receiver's "A" inputs at least 200mV above its "B" inputs. They are also necessary for maintaining fail-safe outputs in the presence of the input-termination resistors. This circuit provides ESD protection, open-circuit detection, and output short-circuit protection, but does not detect short-circuited inputs.

An improved circuit (**Figure 3**) includes two ICs, each of which includes three RS-422/RS-485 receivers. Each receiver provides built-in fault detection, ±15kV ESD protection, and a 32Mbps data rate. The MAX3098E detects open- and short-circuited encoder inputs. It also detects low-voltage differential signals, common-mode range violations, and other faults. Its logic-level outputs indicate which receiver input has the fault condition. By reporting the fault directly, that feature reduces software overhead and minimizes the need for external logic components.

A fault on any encoder input produces an immediate logic high on the corresponding output: ALARMA, ALARMB, or ALARMZ. Slow movement of the servo system can produce transient faults at the quadrature-encoder signal's zero-crossing region, thereby triggering a "false fault." You can delay the ALARMD output (logic OR of ALARMA, ALARMB, and ALARMZ) for a desired interval by selecting the C_Delay value. The 120Ω resistors provide proper RS-422 terminations for the cable. Because the IC is available in a 16-pin QSOP package, this circuit requires fewer components and occupies a very compact space on the PC board.

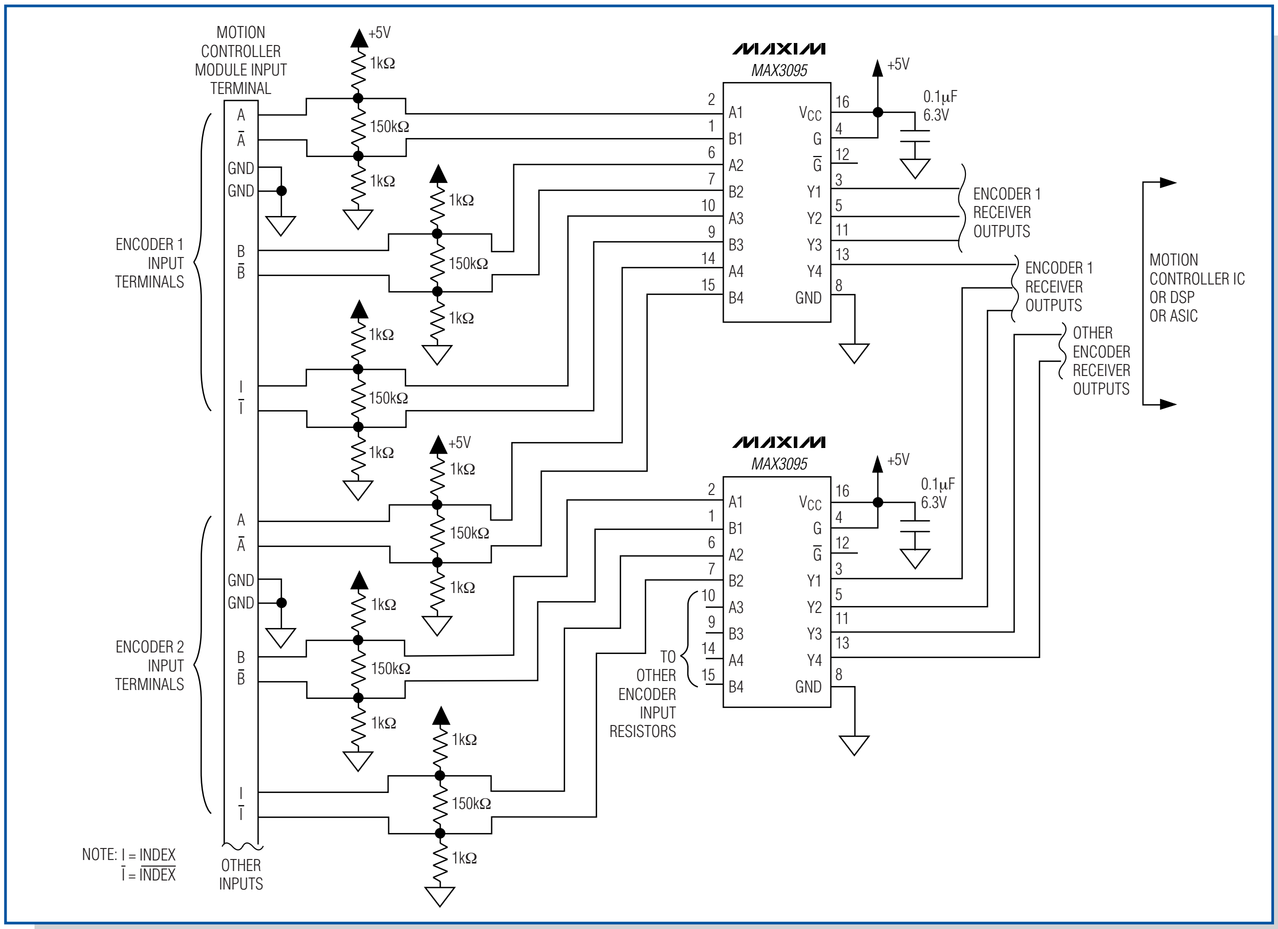
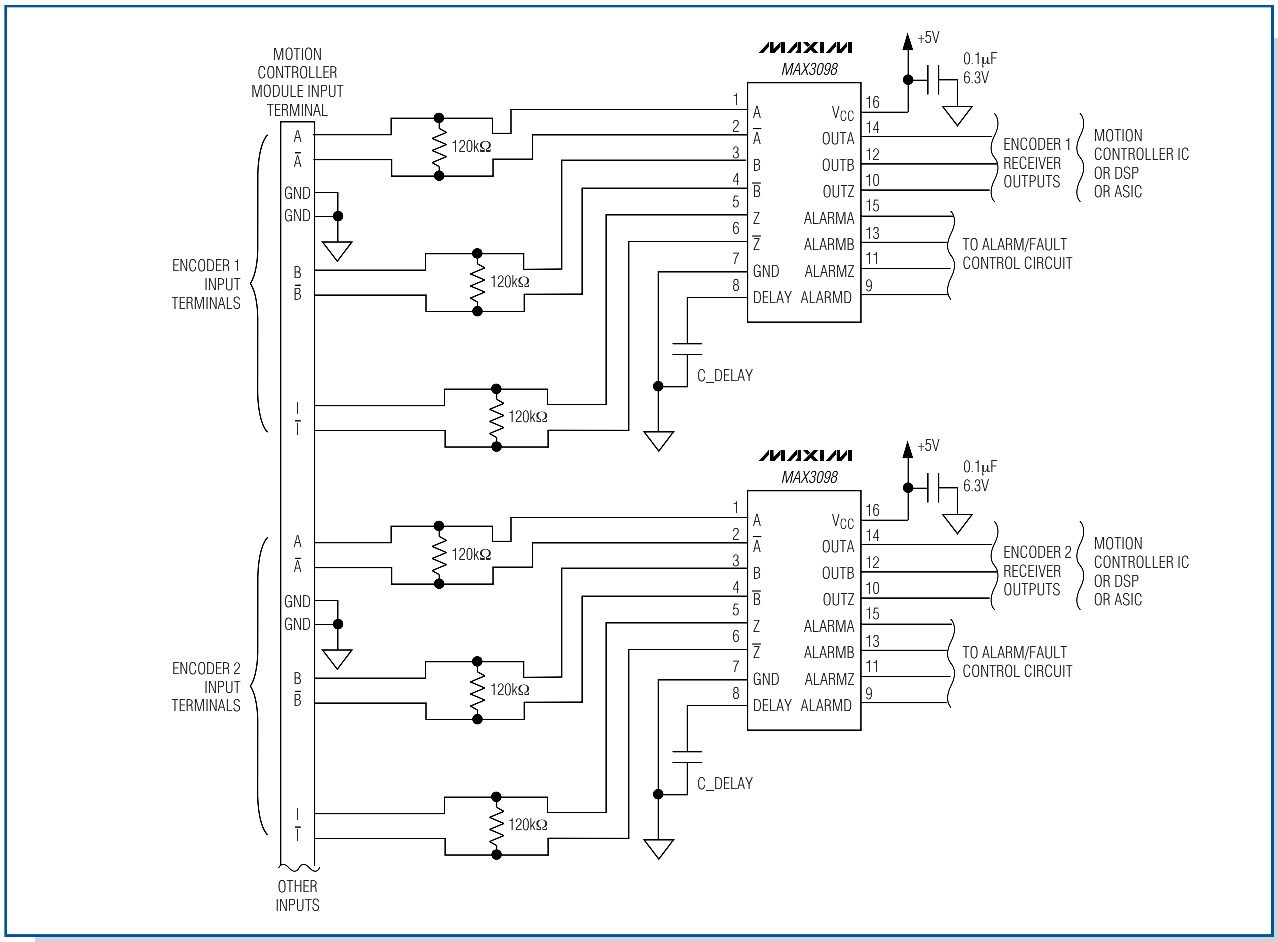

MOTION CONTROLLER MODULE INPUT TERMINAL
+5V
1kΩ
150kΩ
MAXIM
MAX3095
A1
B1
A2
B2
A3
B3
A4
B4
VCC
G
$\overline{G}$
Y1
Y2
Y3
Y4
GND
0.1µF 6.3V
ENCODER 1 INPUT TERMINALS
ENCODER 2 INPUT TERMINALS
ENCODER 1 RECEIVER OUTPUTS
OTHER ENCODER RECEIVER OUTPUTS
MOTION CONTROLLER IC OR DSP OR ASIC
TO OTHER ENCODER INPUT RESISTORS
OTHER INPUTS
NOTE: I = INDEX
$\overline{I}$ = $\overline{INDEX}$

Figure 2. Part of a motion controller, this encoder-input receiver circuit features open-line detection and ESD protection (internal to the MAX3095) on all encoder input lines.



MOTION CONTROLLER MODULE INPUT TERMINAL
ENCODER 1 INPUT TERMINALS
ENCODER 2 INPUT TERMINALS
OTHER INPUTS
MAXIM MAX3098
120kΩ
+5V
0.1µF 6.3V
C_DELAY
ENCODER 1 RECEIVER OUTPUTS
ENCODER 2 RECEIVER OUTPUTS
MOTION CONTROLLER IC OR DSP OR ASIC
TO ALARM/FAULT CONTROL CIRCUIT

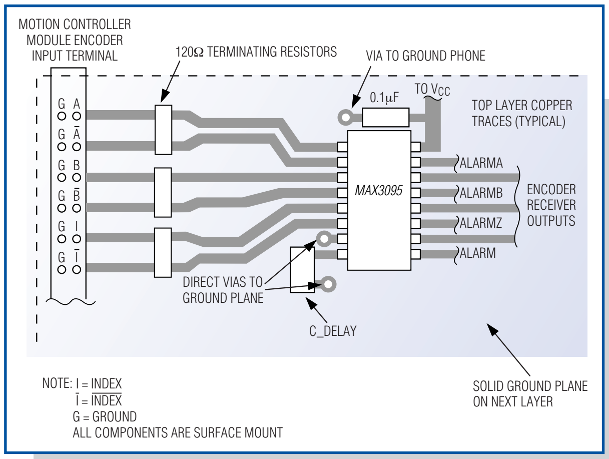
Figure 3. This circuit improves that of Figure 2 with detection of open, shorted, and intermediate faults, ESD protection on all encoder input lines, and delayed alarm/fault outputs.

Receiver circuit PC board layout

A proper layout of the receiver circuit starts with the RS-422 encoder's input connector. The differential signal pairs A A\, B B\, and Index Index\ must occupy adjacent pins on the connector. This configuration minimizes signal imbalance by ensuring that the differential pair's returning signal-current paths overlap and cancel. Typical component placements are shown in **Figure 4**. To ensure that each PC board trace has the same parasitic capacitance, route each differential pair of traces close together, with equal lengths, and with symmetrical bends.

To minimize inductive and capacitive crosstalk on the digital outputs and provide lower inductance, differential RS-422 signals from the connector and the receiver circuit should be laid over a solid ground-plane layer within the PC board. No high-current signals should flow in this ground plane.

The high-speed current switching in motion controller circuits can produce common-mode noise. The use of filters and bypass capacitors helps to reduce the effect of common-mode voltages coupled onto the power supply lines. You should place a 0.1µF bypass capacitor close to



MOTION CONTROLLER MODULE ENCODER INPUT TERMINAL
120Ω TERMINATING RESISTORS
VIA TO GROUND PHONE
0.1µF
TO V_{CC}
TOP LAYER COPPER TRACES (TYPICAL)
MAX3095
ALARMA
ALARMB
ALARMZ
ALARM
ENCODER RECEIVER OUTPUTS
DIRECT VIAS TO GROUND PLANE
C_DELAY
SOLID GROUND PLANE ON NEXT LAYER
NOTE: I = INDEX
Ī = INDEX
G = GROUND
ALL COMPONENTS ARE SURFACE MOUNT

Figure 4. A MAX3098 triple RS-422/RS485 receiver with fault detection ensures proper PC board routing and component placement for the encoder inputs of a motion controller.

the receiver's V_{CC} input. To minimize inductance in the bypass loop, the capacitor's ground lead should connect directly to the solid ground plane, as should the IC ground pin through a via placed adjacent to it. Finally, to minimize noise coupling to the receiver circuits, routing receiver traces near to or adjacent to any power circuits should be avoided.

Encoder signal cable

Because the differential signals from a quadrature encoder are balanced, they can be transmitted on regular paired cable, but twisted-pair cable is preferred. Twisted-pair cables have very low-inductance coupling and a constant impedance up to several megahertz, which enables exceptionally high-speed performance in motion-control systems. Twisted-pair cables also help to reduce radiated and received EMI.

Twisted-pair cable is available shielded or unshielded. Unshielded cable is smaller, costs less, weighs less, and is bendable in a smaller radius, but shielded twisted-pair cable must be used for the differential quadrature-encoder signals. Shielded twisted-pair cable provides better common-mode rejection, because the shield offers additional protection from electric and magnetic interference (EMI). The nonideal twists in an actual unshielded twisted-pair cable allows a dramatic increase in EMI noise. Connect the shield wire to the receiver's ground plane at the encoder-input connector.

The encoder's signal cable should not carry power-level signals or any other signals at all. Nor should it be routed close to or parallel to other cables or conduits that carry power-level signals or other noisy signals, including 60Hz power.

Modern, high-speed servo-control systems operate with encoders that provide data rates up to several megahertz. At such high rates, the encoder-signal cable must be properly terminated with a terminating resistor or network at the receiver end. Ideally, the terminating resistor has the same value as the cable's characteristic impedance.

Because only one transmitter (encoder output) is on the RS-422 network (one transmitter and one receiver), the transmitter does not require a terminating resistor. Ringing and reflections on a nonterminated receiver input, however, can restrict the data throughput to several hundred kilobits per second. Matching the characteristic impedance of a cable to within ±20% is usually more than adequate. Proper termination of encoder cables is illustrated in Figures 2 and 3.

Thus, modern high-speed servo systems can be designed for a robust and fault-tolerant motion-control feedback system. Receiver circuits for the motion controller must respond predictably to the various faults that can develop, and a proper PC layout for the receiver circuit prevents noise problems in the encoder data. A designer should also consider the quadrature encoder's signal-cabling system, including terminations at the receiver circuit. These precautions produce a robust motion-control feedback system that is stable and predictable during fault conditions.

Glossary of terms

Backlash: The mechanical play between two or more adjacent gears.

Index: On a quadrature encoder, the output signal that provides one pulse per revolution.

Latchup: Complete failure in an IC, or momentary loss of operation.

Resolution: The number of bits in an output signal, or (for quadrature encoders) the number of cycles per revolution.

Bibliography

Barnes, John R., *Electronic System Design, Interference And Noise Control Techniques*, Englewood Cliffs, N.J., Prentice-Hall, 1987.

"*New RS-485 IC Increases System Reliability and Fault Detection in Motor-Control Circuits*," www.maxim-ic.com/.

Thomas, Sokira J. & Jaffe, Wolfgang, *Brushless DC Motors, Electronic Commutation and Controls*, Blue Ridge Summit, PA, TAB Books Inc., 1990.

Fault types (FT)

Opens, shorts, and a condition intermediate between the two are the faults most obvious at the system level. Motors and feedback encoders are usually located tens to hundreds of feet from the servo-system controller/amplifier. Connectors terminate these long cable runs at both ends, and it is possible for wires to fall out of the connectors, for connectors to break, and for cables to be inadvertently opened. When end connectors break open or wires sever due to machine vibration, the open or short fault often exhibits several open/reconnect or short/open cycles similar to the contact bounce in a switch before opening up or shorting completely. Because feedback-encoder signals are usually transmitted down a twisted pair, the differential signals are likely to short together during a short fault.

Intermediate faults can develop when the resistance or capacitance of a feedback wire increases, as is possible when a poor installation pinches the cable. Such problems can also manifest during later operation. When moisture enters a damaged cable jacket, for example, the cable capacitance can increase over time, causing signal strength to decrease. That condition is common in heavy industrial environments, where automated equipment can require a daily wash-down. The cable remains operational even though its performance degrades over time. As a precaution, you should include a circuit for detecting moisture-contamination faults.

Noise faults can be the most difficult to eliminate, because noise can originate from electromagnetic interference (EMI), radio-frequency interference (RFI), and/or ground or system-level ground loops. System-level noise sources (radiators) include:

- Arcing of DC-motor brushes during commutation
- High-speed dv/dt switching noise from PWM motor amplifiers
- High-power relays, switches, and actuators such as solenoids
- Random turn-on of SCRs and TRIACs during the 60Hz AC cycle
- Switching power supplies
- Electrostatic discharge

Noise receivers (antennas) include long cables, ground connections, long high-impedance traces on PC boards, and transformers. Noise problems require a method of coupling between a receiver and a radiator, such as capacitive, inductive, or conductive coupling. Capacitive coupling typically occurs in high-impedance circuits when wires or other ungrounded pieces of metal pick up or generate electric fields. For a circuit to couple noise capacitively, the circuit's loop impedance must exceed the intrinsic impedance of air (376.7W).

Inductive coupling typically occurs with low-impedance circuits for which the circuit's loop impedance is less than 376.7W. Wires, open-core inductors, and transformers pick up or generate magnetic fields that can cause EMI noise. The current loops for these circuits must be minimized during design and installation.

Conductively coupled noise usually enters the circuit at ground. For DC noise, it takes the path of least DC resistance in the ground plane, and for AC-coupled noise, the least impedance. As a result, the circuit reference point (ground) tends to be at a voltage above or below its normal value, or (worst case) a dynamically changing value.

Ground loops formed between the AC-power neutral and system-level ground can generate random noise by causing ground current to flow. Ground current can be driven by voltage differences, induction from other cables or devices, wiring errors, ground faults, or the normal equipment leakage that occurs in an industrial environment.

Common-mode noise, defined as common to two nodes that are floating or exhibiting high impedance, can be AC or DC. Common-mode noise can be inherent in the system design, but is usually coupled inductively or capacitively from an external source. A 60Hz signal from the power line, for example, lying adjacent to a pair of signal wires from an analog sensor, can couple inductively onto the wires and drown out the low-level sensor signal.

Electrostatic discharge (ESD) develops when two dissimilar materials come together, transfer charge, and move apart, producing a voltage between them. IC pins that connect to external connectors are susceptible to ESD when a technician connects or disconnects those cables during maintenance.

ESD injected into the pins of an IC can cause the IC to latch up or fail completely. Very high currents can flow during latchup, causing the main power supply to current limit, or causing the system to enter an uncontrolled shutdown. IC pins exposed to external signals or connectors without internal ESD protection must incorporate ESD protection such as metal-oxide varistors, or silicon avalanche suppressors such as TransZorbs. ICs with built-in ESD protection save PC space, and thereby support the drive for smaller form factors and smaller industrial enclosures.

Feedback encoder types (FE)

To attain accurate positioning, a servo system requires a feedback signal to close its feedback loop. Instruments that typically provide the feedback signal include optical encoders, resolvers, and quadrature magnetostrictive linear-displacement transducers. Other instrument types for this purpose are not discussed in this article. They include analog tachometer generators, induction generators, Hall-effect pickups, and potentiomentric devices.

Optical encoders, which provide a digital square-wave feedback signal, include quadrature (incremental), absolute, and pseudorandom types. A typical optical encoder consists of the emitter side, the detector side, and a code wheel, which provides a raw analog signal to the encoder's processing circuitry. A comparator stage then converts the analog to a digital output. Digital formats include open-collector outputs and (for single-ended outputs) 5V–24V logic. For noise immunity, the most robust outputs are the complementary, differential RS-422 types.

Quadrature optical encoders provide feedback signals in the form of A, B, and Z pulses. The A and B signals exhibit a phase separation of 90∞ from the encoder's code wheel and are, therefore, in quadrature (i.e., electrically spaced one-fourth of a period apart). When A goes positive prior to B, the encoder is rotating clockwise, and vice versa for counterclockwise rotation. Thus, position, direction, and velocity data can be derived from these two signals. The Z signal indicates the motor's rotor position, and whether the encoder shaft has rotated 360∞. It also checks for miscounts of the A and B signals. For RS-422 connections, the encoder provides complementary signals for the A, B, and Z outputs.

Absolute optical encoders employ signal-processing components similar to those of the quadrature optical encoder, but their outputs provide one parallel binary word per increment of revolution. Typical outputs are twelve to thirteen bits of BCD, gray, or natural binary code, and the 13-bit outputs impose a lower frequency response (1200 RPM for 12 bits vs. 600 RPM for 13 bits) in exchange for the finer resolution per 360∞ rotation. This encoder type is typically used for monitoring shaft position during power-up and power-down because, unlike quadrature encoders, the encoded output lets you read the shaft position without moving the encoder.

The new pseudorandom optical encoders provide three output signals: A and B provide direction sense and spatial timing, and a third provides position data. Pseudorandom optical encoders require 1-2∞ of rotation to determine position.

Resolvers are feedback encoders that provide sine and cosine output waveforms, which can be processed to provide velocity and position data through the servo controller. A resolver's feedback signals represent absolute position when its shaft is rotating, but low-speed performance is poor. The main disadvantage of a resolver is the relatively expensive resolver-to-digital electronics necessary for processing its signals.

Finally, quadrature magnetostrictive linear-displacement transducers (LDTs) are feedback encoder/transducers designed to measure linear motion, rather than the rotational motion measured by encoders above. The analog position signal is developed from a current pulse sent down a magnetostrictive guide wire, interacting with a position magnet that moves with a linear-displacement rod protruding from the LDT. The reflected pulse is sensed by a pickup sensor. The LDT then processes and digitizes the pickup sensor signal to provide quadrature-output signals A, B, and Z, similar to those of a quadrature encoder.

Finding the middle ground

Developing applications with high-speed 8-bit microcontrollers

Software developers for personal computers have a number of advantages over embedded developers. Not only do they develop for systems that rival the power and memory of supercomputers of just a few years previous, but the systems they develop for generally already exist. Embedded developers, on the other hand, not only have to develop on much smaller systems, but they usually have to design the system first. An approach must be chosen based on the size of the problem. If the problem has little user interaction, controls a small number of devices, and is a relatively simple design, it can be tackled with a low-power, 8-bit microcontroller, like an 8051, 68HC11, Amtel AVR, or PIC variant. These usually provide adequate power and flexibility. If the problem has large amounts of user interaction, needs to talk over Ethernet, or needs to talk to complex devices like digital cameras, then usually a PC-104, StrongARM, or another type of "one-calorie personal computers" is used. These generally provide abundant processing power, complex operating systems, and large amounts of RAM.

There is a gray zone in between. For example, let's say Joe's Security Service is entering the cutthroat market of network door locks. Due to recent security alerts, companies want to install door locks that log users with more than just an ID number. They want electronic door locks that take a photo of either the person's face or their thumbprint whenever the user wants to open the door. These images are sent over a network to a central server either for logging or, in a complex example, image recognition and validation. If the image is validated, the server sends back a response to the network door lock, and the door opens for the user. Joe would like to have his customers install many doors throughout their facilities, so it is important to keep costs low.

One of Joe's competitors, Alex's Security Central, is developing a network camera door lock using an 8-bit RISC connected to an Ethernet controller. Joe dismisses this solution as underpowered. He knows that there have been a number of projects involving connecting these Harvard-architecture chips to an Ethernet controller. However, many of these projects are in their infancy, none of them are commercially supported, and the TCP/IP stack is limited by the architecture itself. If talking over the network did not disqualify them, talking to a digital camera would. An adequate image would require 40kB to 60kB of memory, which has to compete with the code memory space as well. Even if he was using something with non-Harvard architecture, there is just too much work to be done and data to be processed with a traditional 8-bit microcontroller.

TINI and 1-Wire are registered trademarks of Dallas Semiconductor.
SPI is a trademark of Motorola, Inc.

Joe's arch nemesis, Troy of Troy's X-Treme Security, is also developing a solution. Joe hates Troy because Troy has no respect for the art and finesse of embedded system design. For romance's sake, we will also say that Troy is dating Joe's ex-girlfriend, Amiga, who left Joe because he spent too much time at the computer (also known as an "occupational hazard" among embedded system developers). Troy is developing a solution using a StrongARM running Pocket PC, which has speedy I/O and networking capabilities. Joe sees this solution as overkill. Beyond taking a photo, the processor will sit idle most of the time. The ideal solution does not need a lot of memory or power, so running embedded Linux or Pocket PC would add unnecessary bloat, and too much cost, for such a simple device.

What Joe needs is a microcontroller powerful enough to handle the network and the camera, but less expensive and functional than a 32-bit solution. It would help Joe if it supported a higher level language than pure assembly in order to simplify development. How will Joe be able to beat Alex's power, Troy's price, and win back his true love?

Enter TINI®.

A network bridge

TINI, or tiny Internet interfaces, is a product of Dallas Semiconductor. The TINI platform is designed to work as a network bridge: a PC can talk to TINI over TCP/IP, and TINI talks to a sensor, legacy hardware, or other device. TINI offers a number of external interfaces, including 1-Wire®, 2-wire, RS-232 serial, CAN, and SPI™. TINI has a robust networking implementation, and offers support for IPv4, IPv6, DNS, DHCP, PPP, Telnet, and FTP.

There are two reference designs available. The most common, the TINIm390 verification module based on the DS80C390, is a 72-pin SIMM that includes 512kB of flash, 512kB or 1MB of battery-backed SRAM, an Ethernet controller, and a real-time clock. The new version, the TINIm400 verification module based on the DS80C400, is a 144 SO DIMM with similar features, except the Ethernet MAC is built into the DS80C400.

The 390 and the 400 are both 8-bit microcontrollers. In fact, they are both 8051 microcontrollers at the core. However, they have been heavily expanded from their original roots. First, their cores are 4 cycles-per-instruction instead of the standard 12. This gives a triple-speed increase over standard 8051s at the same clock frequency. Second, they have much higher addressing capabilities. The 390 supports 4MB of program memory and 4MB of data memory, and the 400 supports a flat 16MB address space. Third, they support much higher clock frequencies. The 390 can run up to 40MHz, while the 400 can run up to 75MHz. Lastly, they both have integer math accelerators for multiplication and division. They offer a middle range of power between a traditional 8-bit microcontroller and a 16-/32-bit microcontroller.

A unique feature of the TINI platform is the operating system developed by Dallas. It is a royalty-free, multi-tasking, multithreaded operating system that boasts a Java™ runtime environment and is available as a free download. The core OS and libraries fit into a 512kB flash with enough room for a 64kB application in the last flash bank. The DS80C400 also contains a ROM library for C and assembly programming.

Network camera

To show what is possible regarding the problem of the network camera, the TINI is used to implement a streaming web camera and to benchmark performance. Rather than use the TINIm400 reference design, a custom, high-speed DS80C400 design is used (**Figure 1**). The network camera discussed here will take raw images and send them over UDP to a PC host. It will talk to the PC either using host-side software, or through a Java applet served over HTTP.

The camera chosen is the M4088 module, which uses an OmniVision 5017 CMOS chip. The camera is a noninterlaced, black and white digital camera with a 384 x 288 pixel resolution. The camera exposes 8 data lines, 4 address lines, and the chip select, allowing it to be memory mapped easily. It also exposes a vertical sync line that asserts when an image is being taken, a horizontal sync line that asserts for every scan line, and a pixel clock that informs when a pixel is coming. Once the camera is initialized, accessing it from software is easy. The 5017 has an internal clock divider that allows programmatic control of the frame rate. It also has a single frame mode, which allows the host device to control when an image is taken. This is useful, since this design does not have the processor power to handle the camera's top speed of 50 frames/s.

Java is a trademark of Sun Microsystems.
Purchase of I2C components of Maxim Integrated Products, Inc., or one of its sublicensed Associated Companies, conveys a license under the Philips I2C Patent Rights to use these components in an I2C system, provided that the system conforms to the I2C Standard Specification as defined by Philips.

The 400 version of the camera is designed to run at 73MHz (18.4MHz x 4). The 400 has a built-in Ethernet MAC, but it needs an Ethernet PHY and magnetics. It supports many other PHYs, such as HomePNA and HomePlug PHYs. An Intel LXT972A is used in this example, which connects directly to the 400 using a standard MII interface (**Figure 2**). The PHY requires its own 25MHz clock.

The camera must have 12ns memory for execution. A Hitachi HM62W8511H is used, which provides the necessary access time. On boot, the processor executes bootstrap code from flash that copies the executable image from flash to SRAM, flags the clock quadrupler to be enabled, and jumps to the TINI starting address. The board does not have the battery backup or nonvolatizer on this configuration that is available on the TINIm400, as the high-speed SRAM would drain the battery too quickly. This means that TINIOS does not have a persistent file system. This is not as much of an issue as it seems since, as can be seen later, TINI will build the file system on startup.

TINIOS requires a DS2502 for MAC address storage. While not required, there also is a DS1672 real-time clock connected over I2C™. This gives TINI access to the clock in software, but provides an additional bonus feature. On boot, if a real-time clock is detected, TINIOS autocalculates the system bus speed and adjusts its internal timers accordingly, including serial port baud rates, timer ticks, and network timings.

Connecting the camera is straightforward; A0–A3, D0–D8, and WEB hook to the appropriate lines. The CE4 is connected to the camera's CSB line, which maps the camera at 0 x 800000. The VSYNC line of the camera is connected to the P1.1 line for reading, and PSEN to the OEB camera line. The HREF line must be inverted before connecting it to INT1 so that level-triggered interrupts can be used.

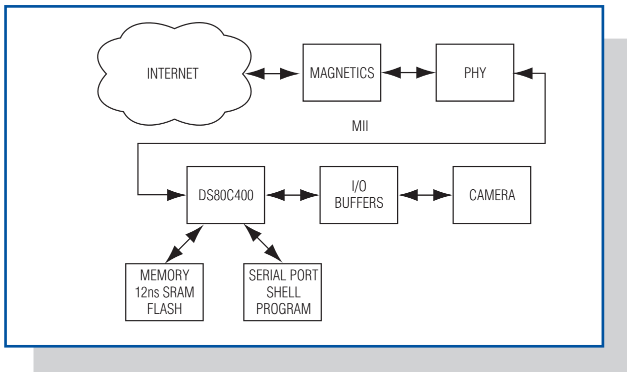


Figure 1. This diagram illustrates the design of the high-speed DS80C400.

Software implementation

For development, the TINI SDK is used, which is available as a free download from the Dallas Semiconductor website. One might assume that a Java virtual machine would not have sufficient power to capture the camera data fast enough. However, TINI allows for interrupt service routines to be installed under the operating system that communicates with the application using native methods. This shows one of the strengths of the TINI platform; high-level protocols (like HTTP) can be implemented in pure Java, while high-speed portions of code can be implemented with native 8051 assembly. In many ways, it is the best of both worlds.

TINI supports the following packages from JDK 1.1:

- java.io
- java.lang
- java.net
- java.util
- javax.comm

Some differences do exist between TINI's JVM and a PC's JVM. First, while TINI supports garbage collection, it does not support finalizers. This means the programmer is required to close their resources explicitly instead of having it done by the system, but this kind of resource management is a good idea in embedded development anyhow. TINI also has some size limitations on classes: any individual class file cannot be larger than 64kB, a method can only have 63 locals, and arrays can only be 64kB in length. The system is also limited to 8 processes, and 32 threads per process. Not all the differences are limitations, though. For example, the 1.1 JDK does not have any support for IPv6, so the TINI implementation comes from the 1.4 JDK.

In addition to the standard Java classes, TINI provides the following packages:

- com.dalsemi.comm
- com.dalsemi.fs
- com.dalsemi.system
- com.dalsemi.tininet

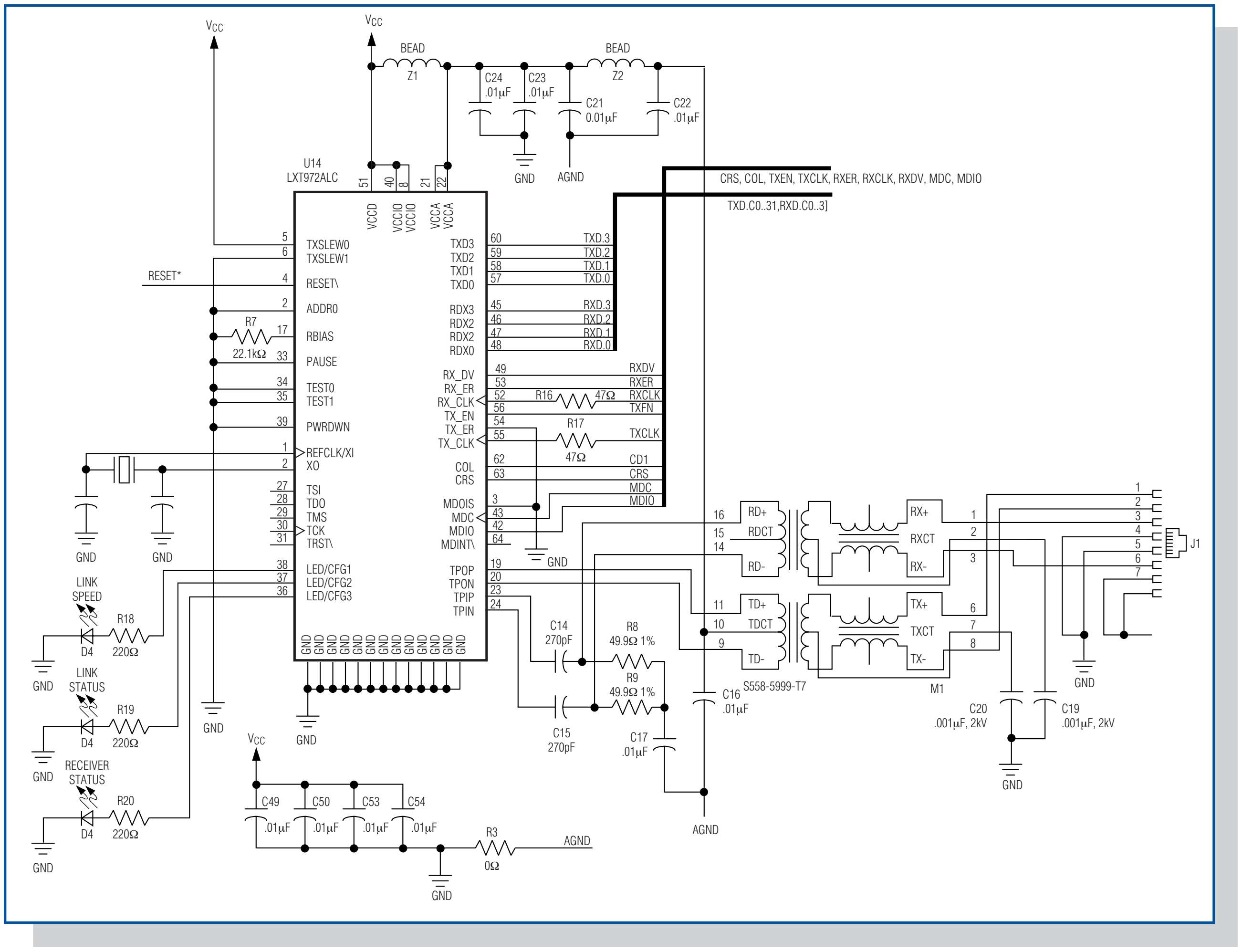


Figure 2. The DS80C400 connects to the PHY using a standard MII interface.

These packages provide both low-level system access and support for other protocols, like CAN and I^{2}C. TINI also has support for a number of high-level protocols. For example, one of the examples in the TINI SDK is a lightweight (3kB .class file) HTTP server.

The software follows the design outlined in **Figure 3**. At the bottom layer is the camera interrupt service handler, which holds a persistent pointer to the camera buffer in indirect memory. The camera is set to single-frame mode, where it waits for a command before transmitting the image. Once flagged, the image is transmitted synchronously at a rate controlled by the FRCTL register. On the 400, the camera has been set to a rate of 10 frames/s, which transmits a 384 x 288 image in 1/10th of a second at a transfer rate of 1080kB/s.

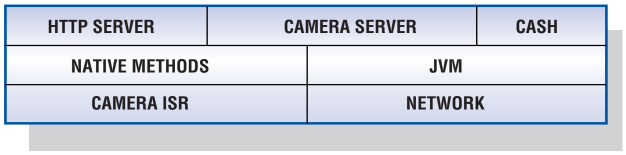


Figure 3. TINI streaming camera software supports both low-level and high-level protocols.

Talking to the camera using an 8051 assembly is relatively easy. Each pixel of the image is read synchronously from one camera register. Since the camera is memory mapped, reading and writing from its registers involves pointing the data pointer at the camera address and executing a movx opcode. In a traditional 8051 assembly, moving from one address to another would involve loading the data pointer, reading memory into the accumulator, setting the address to have it copied to the data pointer, and writing the accumulator to memory.

```
  mov   R0,#LOW(MEMORY_LOW)
  mov   R1,#HIGH(MEMORY_HIGH)
camera_loop:
;
; Move the camera address into the data pointer.
;
  mov   dptr,#CAMERA_ADDRESS
;
; Move the data into the accumulator.
;
  movx a,@dptr
;
; Move to the address we will be writing to. Since
; this will increment every time, we will keep
it stored
; in registers. We will also need to move it one
byte at
; a time using the DPL and DPH SFRs.
;
  mov dpl,R0
  mov dph,R1
;
; Write the accumulator to the address
  movx @dptr,a
;
; Increment the data pointer and store back in
R0 and R1.
;
  inc dptr
  mov R0, dpl
  mov R1, dph

; Do the loop again...
```

The DS80C400 has four data pointers, allowing for data to be quickly copied from one address to another. This removes all the address swapping, making a copy run much faster.

```
;
; Set up the data pointers. We use the DPS
register to select
; what data pointer we want to use. A data
pointer move allows
; for a 24-bit address to be loaded directly.
;
  mov dps,#0
  mov dptr,#CAMERA_ADDRESS

  mov dps,#1
  mov dptr,#MEM_ADDRES

;
; Set data pointer 0 as the current data pointer.
  mov dps,#0
camera_loop:
;
; Read from data pointer zero.
;
  movx a,@dptr
;
; Switch to the next data pointer. Note that doing
; an inc on this register only affects the data
; pointer-selection bit. This allows one cycle
toggling
; from one data pointer to another.
;
  inc dps
;
; Store the data and increment the address.
;
  movx @dptr,a
  inc dptr
;
; Switch back to the first data pointer.
;
  inc dps
;
; Do the loop again...
;
```

As can be seen, the loop runs faster because most of the address handling is done outside of the loop. For memory copies, the DS80C400 also has optimizations for even faster copies. First, all data pointer increments are performed in a single cycle. An autoincrement mode, when enabled, can automatically increment the address in the data pointer after a read or write is performed. Also, an autoselection feature can be enabled to toggle between two data pointers with every read or write operation. This makes a memory copy incredibly easy.

```
;
; Set the base address of data pointer zero.
;
  mov dps,#0
  mov dptr,#ADDRESS1
;
; Set the base address of data pointer one.

  mov dps, #1
  mov dptr,#ADDRESS2

;
; Enable autoselection and autoincrement.
;
  mov dps, #(DPS_AID | DPS_TSL)
memory_loop:
;
; Read from data pointer 0, increment the
; data pointer, and toggle the selection bit
; in one instruction.
;
  movx a,@dptr
;
; Write to data pointer 1, increment the data pointer,
; and toggle the selection bit in one instruc-
tion.
;
  movx @dptr,a

; Loop...
```

Back to the example, the HSYNC line is used to generate the interrupt. When the camera asserts the HSYNC line on each scan line, the ISR fires and begins to capture data. In order to improve performance, all other interrupts have been moved to low priority, including the scheduler. This has a noticeable effect on image quality.

The camera has a programmable window which allows the user to configure how much of the 384 x 288 image to use. Finding an appropriate image size is difficult; a large image has increased quality, but takes more time to transmit and reduces frame rates. For this application, Joe sets the camera to a 240 x 180 resolution. This is a standard resolution for Internet video, and has an additional hardware advantage. Looking at **Figure 4**, when the camera is transmitting the image, it actually enumerates every pixel the image array, but only asserts the HSYNC line for pixels inside the specified window. This means that an image capture consumes about 3/5 of the CPU during the 100ms period.

On initialization, the camera software allocates a 95kB contiguous block of memory from the memory manager,

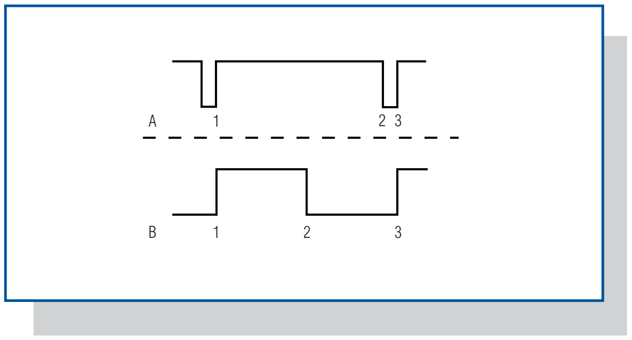


Figure 4. The HSYNC line is asserted for each individual scan line. In A, the pixel window has been set to the full 384 pixels, while in B it has been set to 192 pixels, beginning at the left edge with both set to the same frame rate. Marker 1 is where the scan line is asserted by the camera, marker 2 is where it finishes with the scan line, and marker 3 is the time when the camera starts the next scan line. Though B takes half as long as A, the period per scan line is the same.

which is used for double buffering. In Java, one thread handles all of the image capture, while another thread handles transmitting the image over the network. Java simplifies this by providing all of the threading and locking mechanisms necessary.

Above the ISR in the software design are the native methods that provide the camera driver for the Java virtual machine. The TINI platform uses the TINI native interface (TNI) to allow developers access to low-level code from Java. Native methods can call into the TINIOS native API, allowing them access to the memory manager, scheduler, and other operating system internals. They can have parameters passed to them, and even throw exceptions like other Java methods. These are linked to the Java executable through a TINI dynamically linked library (TLIB), which can be built using the TINI developers kit.

The native methods allow Java to send commands to the camera. The camera driver class has the following definition:

```
public static final int IMAGE_BUFFER_0 = 0;
public static final int IMAGE_BUFFER_1 = 1;


/**
takePhoto takes a photograph and stores it in
the memory.
@param buffer Species what image buffer to use.
Use IMAGE_BUFFER_0 or IMAGE_BUFFER_1
*/

static native void takePhoto(int buffer);

/**
getScanlines pulls a fixed number of
scanlines out of the memory buffer. This
allows the Java application
```

```
  the ability to work with fixed pieces of the
image.

  @param start first scanline to copy from.
  @param end last scanline to copy from
  @param offset offset into the data array to
copy to
  @param data array to copy into
  @param buffer selects the image buffer to read
from. Useful IMAGE_BUFFER_0 or IMAGE_BUFFER_1
  */
  public static native int getScanlines(int
start, int end , int offset, byte []data, int
buffer);
```

The main method, takePhoto, captures one image into the image buffer. First, it enables the interrupt, and sends the camera a command to take a single image. There is a small impasse here. It is preferred that the Java thread is put to sleep at this point; however, it cannot be woken up from the ISR because the TINIOS functions are not reentrant. For this purpose, TINIOS allows developers to register poll routines that are called every 4ms by the system. A poll routine does a quick check to see if the photo is finished, and wakes up the thread if it is. This poll routine is registered right before putting the thread to sleep. Upon waking up, the thread returns to Java. As mentioned previously, the camera is double buffered, so the image buffer to overwrite must be specified. In order to let Java access the underlying image buffer by allowing blocks of scan lines to be copied into Java byte arrays, getScanlines is also implemented.

A storage problem is also present. The TINI runtime takes 7 banks of a 512kB flash, leaving one bank for a user application. As mentioned before, there is not a nonvolatile file system on the high-speed design, so the file system must be created from scratch. It is desirable to have everything necessary to run from power-up in the flash bank, including the Java applet that the HTTP server will serve to the web browser. In order to include the applet in the executable, the applet binary is converted into a format compatible with the assembler and included in the data segment of the library. Then there is a native method to copy it from the library into a Java byte array. On startup, the Java code reads the size of the applet, creates an array, copies the applet into the array, and writes the contents to the file system. It is a little clunky, but it means one can start from a clean boot and have everything necessary to start. The following methods perform this task:

```
  /**
Extracts the sample jar file from the native
library. The demo application had a jar file
embedded inside the native library.
This allowed the jar file, the application, and
the native library all to be embedded in flash
format.
  @param dummy Array to copy jar image into.
Must be of greater size than that specified in
getJarFileSize()
  */
  static native void getJarFile(byte []dummy);

  /**
  Gets the size of the embedded jar file
  */
  static native int getJarFileSize();
```

The Java on top of the camera driver is much simpler. A number of threads are running, most of them independent from one another. First, there is the HTTP server. The code for this comes from the HTTP server example included with the TINI SDK. It is very lightweight, and is not designed for servelets, cgi-bin processing, or other features. Files are read from the TINI file system, which is a hierarchical file system implemented in TINI's nonvolatile memory. On startup, the camera applet is read from flash memory and written to the file system, and the index.html page is generated.

Next, there is the camera image server. The camera server has two main threads. The first thread opens a TCP server socket on port 42877 and awaits connections from an applet. How is server socket opened on an embedded system? Actually, it is much like how it would be done on a PC.

```
  sockpuppet = new ServerSocket(42877);
```

Something to note is that a server socket binds to a port on both the IPv4 and IPv6 interfaces. This means that no changes are necessary to make the camera IPv6 compliant. IPv6's much larger address space will help network appliances in the future, since they currently need to fight for addresses with PCs, cell phones, and other devices on a network.

When a process connects, it sends either an 'A' for connect or 'D' for disconnect. On a connect command, the IP address is added to a shared vector of connected addresses, while the disconnect command removes it from the vector.

```
  sock = sockpuppet.accept();
  ch = (char)sock.getInputStream().read();

  switch (ch)
  {
  case 'A':

  cwt.addAddress(sock.getInetAddress());
  break;
  case 'D':

  cwt.removeAddress(sock.getInetAddress());
  break;
  }

  sock.close();
```

The other thread is the image transmitter. If there is an address in the camera vector, it transmits the captured image to it in UDP packets. The camera capture and transmission have been optimized to run in parallel. The capture has been double buffered to allow transmission from one buffer while capture occurs in another. This is possible because the camera only uses about 50% of the CPU while transmitting. The asynchronous locking is all done in Java.

```
  //
  // Notify the camera thread we are ready for
  // the next frame.
  //
  synchronized(stopper)
  {
        stopper.notify();
  }
```

No sort of image compression hardware is utilized, so the image is transmitted raw. The packet layout is extremely simple. Each packet has a 2-byte header, followed by the data for five scan lines. The first byte is the frame number, which is a rolling counter that increments for every frame transmitted. The second byte is the vertical offset divided by five.

Finally, a configuration tool is run on the serial port. This application, the CAmera SHell or CASH, is a menu-driven utility that allows the IP address to be set, and the connected users to be seen. A lot of this functionality is taken from the Slush shell that comes with the TINI SDK. To configure the camera, the user powers it up and communicates with it over the serial port. The CASH presents a simple user interface for setting the IP address. Once configured, it can simply sit on the network, waiting for a user. When someone connects with a web browser, the camera serves the applet, which makes the connection to the camera server and displays the images. While TINIOS supports up to 24 simultaneous open sockets at a time, for speed reasons we limit the camera to four users at a time. Using multicast would alleviate this problem, but Java applets do not support it.

The network camera captures and transmits 4.5 frames per second, with an average network transmit rate of 200kB/s. Something to remember is that very little support hardware has been connected to the camera. There is no image capture or encoding hardware in the system, which means the DS80C400 is juggling image capture, image transmitting, network traffic, web serving, and user interaction over the serial port at the same time.

Conclusion

Let's return to Joe. Seeing that TINI is the ideal solution for his product, he continues his development of the network door lock. With his low-cost, high-power solution, Joe becomes the reigning king of the network door lock market. He easily beats Troy's solution that, despite bearing the logo of a international software monopoly, just costs too much. Alex's solution, having to overcome both camera and network issues, has difficulties making it to market. Amiga leaves Troy and returns to Joe, who promises her that he will never again let the computer come between the two of them. With the success of his business, Joe and Amiga retire to a small cabin in Minnesota that, of course, has a network door lock on every door.

Embedded devices need to be designed to solve specific problems. It is a challenge to find the right balance between power and cost. This becomes even more complicated when adding network capability to a device. One route is to try to extend an 8-bit microcontroller into the networking world. While it is feasible, it will inevitably be slow. Another approach is to use an embedded Linux, PC-104, or Pocket PC device. While this will be fast and responsive, it also adds a lot of unnecessary bloat. One could build a smaller 32-bit solution, but that requires licensing an operating system and TCP/IP stack.

The DS80C400 with TINI is a good in-between solution. It has a robust, well-rounded TCP/IP stack that has been hardened over time. It has an operating system with support for multiple processes, Java, threading, and synchronization. The processors can handle heavyweight tasks like talking to a digital camera without the bloat of a heavyweight operating system. If it works for the average Joe, it might work for you.

Do passive components degrade audio quality in your portable device?

For an audio circuit, passive components define the gain, provide biasing and power-supply rejection, and establish DC-blocking from one stage to the next. The limitations of portable audio, in which space, height, and cost are usually at a premium, force the use of passives with small footprints, low profiles, and low cost. The audible effect of these devices is worthy of some examination, because poor component choice can significantly degrade the measured system performance.

Some designers assume that resistors and capacitors have no measurable effect on audio quality, but the nonlinear characteristics of many common passives used in the audio signal path can seriously degrade total harmonic distortion (THD). In some cases, the nonlinear contribution of passives exceeds that of active devices such as amplifiers and DACs, which are assumed by many designers to be the limiting factor in audio performance.

Sources of nonlinearity

Capacitors and resistors both exhibit a phenomenon called voltage coefficient, in which a change in voltage across the component changes its physical characteristics and hence its value, to some degree. For example, a particular 1.00kΩ resistor with no voltage across it becomes a 1.01kΩ resistor when 10V is applied. That effect varies enormously according to the component's type, construction, and (for capacitors) chemistry. Voltage-coefficient information is sometimes available from the manufacturer as a graph, showing the percent change in capacitance vs. the percent of rated voltage.

The voltage coefficient of modern film resistors is very good, and usually below the level that is readily measured in the lab. Capacitors, on the other hand, can limit performance as a consequence of several departures from the ideal:

- *Voltage coefficient:* Described previously.
- *Dielectric absorption (DA):* A memory-like effect in which a discharged capacitor retains some of the charge previously stored on it.
- *Equivalent series resistance (ESR):* This can be frequency dependent, and can limit power output when series-coupling capacitors drive the low impedance of a headphone or speaker.
- *Microphony:* Some capacitors have a marked piezoelectric effect, in which physical stress and flexure of the capacitor generates a voltage across the terminals.
- *Poor tolerance:* For most large-valued capacitors (several µF and higher), accuracy is not tightly specified. Resistors, on the other hand, are readily and inexpensively specified for a tolerance of 1% or 2%.

The following discussion outlines a test method consisting of a simple test circuit and readily available audio test equipment that can quantify the undesired effects of capacitors in the audio signal path. The intent is not to pass judgement on particular sizes, voltage ratings, and case types, but to alert readers to the phenomena, show representative results, and offer a test method that allows meaningful comparisons and conclusions.

Test description

Nonlinear AC effects are easily found in capacitors. The frequency response of analog audio (necessarily restricted) dictates that most circuit blocks include highpass, lowpass, or bandpass filter circuits, and that the nonlinearities of such filters can have real and measurable effects.

Consider a simple, highpass RC filter (**Figure 1**). At frequencies well above its -3dB cutoff, the capacitor's impedance is low with respect to that of the resistor. The application of a high-frequency AC signal develops very little voltage across the capacitor, so any change due to the voltage coefficient should be minimal. Signal current flowing through the capacitor, however, generates a corresponding voltage across the capacitor's ESR. Any nonlinear component of that ESR sums in at the appropriate level and can degrade THD.

At and near the -3dB cut-off point, however, impedances of the capacitor and resistor are of the same order. The result is a significant AC voltage across the capacitor at a point in the response that imposes only minor attenuation on the input signal. Thus, any voltage-coefficient effects tend to peak around that point.

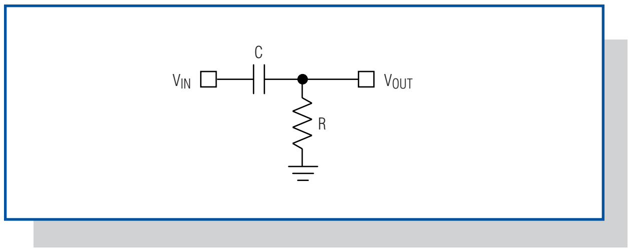


Figure 1. Simple highpass RC filter.

By focusing on THD at the -3dB cutoff, this test highlights nonideal behavior—primarily that due to the voltage coefficient. The test circuit is based on a highpass filter with -3dB cutoff at 1kHz and an audio analyzer (Audio Precision System One) that looks for any degradation of THD+N while various capacitors of differing construction, chemistry, and type are substituted. A 1µF capacitor value was chosen because it offers a wide choice of capacitor types for testing. It is loaded with a 150Ω resistor to form a headphone filter with nominal 1kHz cutoff. Note that the capacitor under test has no DC bias across it. Input and output have the same DC potential.

Polyester capacitor and reference baseline

A plot of THD+N vs. frequency in **Figure 2** shows the limit of resolution in the test setup, and also the minimal effects of a 25V through-hole polyester capacitor not typically used in portable designs. Little, if any, THD degradation due to voltage coefficient is apparent. Note that the polyester capacitor allows THD to rise below 1kHz, but the output signal is actually falling with a frequency below 1kHz, thereby reducing the ratio of signal-to-noise (plus distortion) registered by the analyzer. The key region is at and above 1kHz, where the polyester capacitor performs well—only slightly worse than the reference measurement.

Tantalum dielectric

Tantalum capacitors are often found in portable devices, usually for blocking DC voltage to a headphone and especially if more than a few µF are required. Another plot of THD+N vs. frequency in **Figure 3** compares three variations of a popular SM tantalum capacitor with a traditional, through-hole "dipped" tantalum capacitor readily available in the lab. The capacitors again have 1µF values; only their physical dimensions (case size) and voltage ratings differ. See **Table 1**. No DC bias was applied to the capacitors during this test.

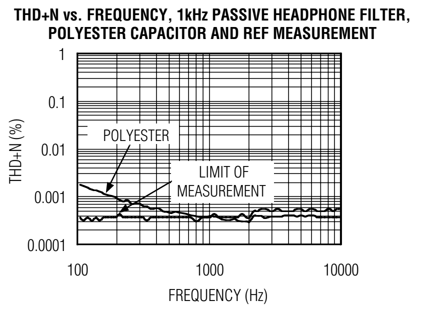


Figure 2. THD+N vs. frequency for a 1kHz highpass passive filter with polyester capacitor, compared to a reference measurement.

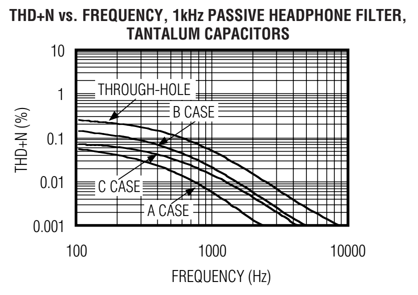


Figure 3. Comparison of THD+N vs. frequency for various tantalum capacitors in a 1kHz highpass passive filter.

Ceramic dielectric

Ceramic capacitors are often used for AC-coupling between audio stages, and in bass-boost and filtering circuits. Various dielectric types are available, as **Figure 4** illustrates, based on the components listed in **Table 2**.

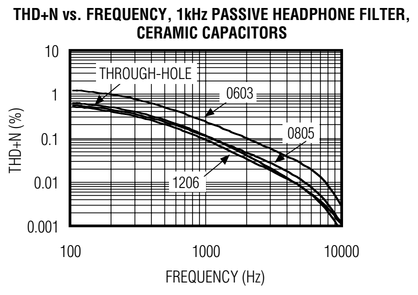


Figure 4. Comparison of THD+N vs. frequency for various ceramic capacitors in a 1kHz highpass passive filter.

Table 1. Comparison of SM tantalum capacitors tested in Figure 3

Value	Case Size L x W (mm)	Voltage Rating (V)
1µF	A (3.2 x 1.6)	25
1µF	B (3.5 x 2.8)	35
1µF	C (6.0 x 3.2)	50

Table 2. SM ceramic capacitors tested in Figure 4

Value	Case Size	Voltage Rating (V)	Dielectric Type
1µF	0603	10	X5R
1µF	0805	16	X7R
1µF	1206	16	X7R

Figure 4 also depicts the performance of a through-hole ceramic capacitor selected from a random assortment of lab components. The worst result is just 0.2% THD at the -3dB point for the X5R dielectric. To put it in perspective, that performance equates to distortion at the -54dB level. The THD for most 16-bit audio DACs and CODECs, with respect to full scale, is at least an order of magnitude better than this. Note that C0G dielectrics can exhibit very low voltage coefficients, but at this time their capacitance ranges are restricted to values near 0.047µF and below. These tests were based on 1µF capacitors, so C0G types were not included.

How to avoid capacitor voltage-coefficient effects

Figure 5 shows a line-input topology whose novel AC-coupling configuration allows a much lower valued input capacitor than that of a traditional configuration. The input capacitor in this example (C1) is 0.047µF, which can be specified as a ceramic with C0G dielectric in a 1206 case size—a configuration that minimizes the THD contribution from voltage-coefficient effects. DC feedback for the op amp (which should be a device with low input-bias current, such as the MAX4490) is provided by the two 100kΩ resistors. The effect of the DC-feedback path at audio frequencies is attenuated by C2 and R5, so the majority of the feedback is from R1 and R2 through C1. With the values shown, the -3dB point is set at 5Hz.

This type of compound feedback ultimately has a first-order LF rolloff, but can be tuned for a 2nd-order response around the highpass cutoff frequency. Consequently, pay careful attention to overshoot and peaking when adjusting the component values from those shown in **Figure 6**. Values in the example approximate a maximally flat highpass function. This circuit principle can easily be adapted to quasidifferential (ground-sensing) and fully differential input stages.

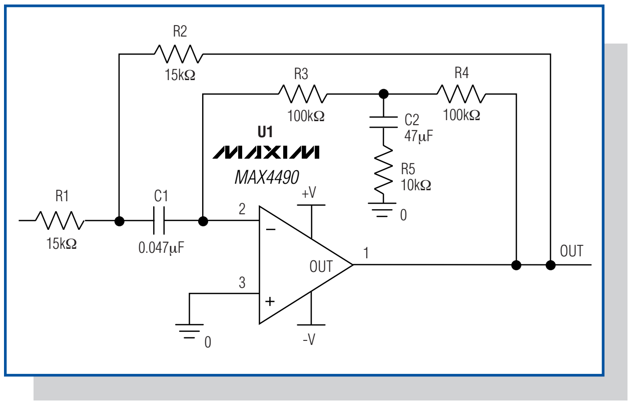


Figure 5. This novel line-input stage reduces degradation due to voltage-coefficient effects. Including the traditional AC-coupling capacitor inside the amplifier's error path lowers the value of that capacitor, and enables the use of C0G capacitors in portable designs.

The stereo headphone driver IC shown in **Figure 7** (MAX4410) features an innovative technology called DirectDrive, in which the output bias, powered from a single positive supply, is set at 0V to allow direct DC-coupling to the headphones. Several advantages follow:

- Large DC-blocking capacitors (100µF–470µF, typ) are eliminated, which removes a significant THD contribution based on voltage coefficients.
- The lower -3dB cutoff, now defined by the input capacitor and input resistor, is around 1.6Hz with the values shown in Figure 7, but a -3dB point of 1.6Hz in standard AC-coupled headphone drivers for 16Ω headphones requires about 6200µF. In addition, the low-frequency response is no longer load dependent.
- Eliminating the large-case capacitors saves a significant amount of PCB area. Such capacitors are expensive when compared with the 1µF and 2.2µF ceramic compacitors required by MAX4410 charge-pump circuitry.
- To enable the outputs to sink and source load current with a ground-referenced load, the chip generates an internal negative supply for the amplifier. Because that supply (PV_{SS}) is an inverted version of the positive supply (V_{DD}), the available voltage swing at the output (almost $2V_{DD}$) is twice that of a traditional single-supply, AC-coupled headphone driver.

In this example, it has been a relatively simple matter to minimize the voltage-coefficient effect of input capacitors on audio bandwidth by oversizing those capacitors. Given a 10kΩ input resistor, choose a 10µF ceramic for C_{IN}. That combination places the -3dB point at 1.6Hz, so the worst effects due to voltage-coefficient nonlinearity are at least an order of magnitude below the lowest audible frequencies being reproduced.

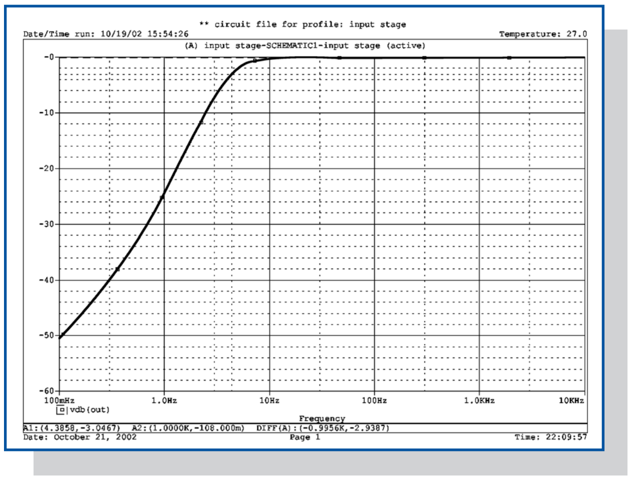


Figure 6. Frequency response for the circuit in Figure 5 shows a smooth rolloff below 10Hz with the -3dB point at 5Hz. The ultimate rolloff rate with decreasing frequency is 20dB/decade.

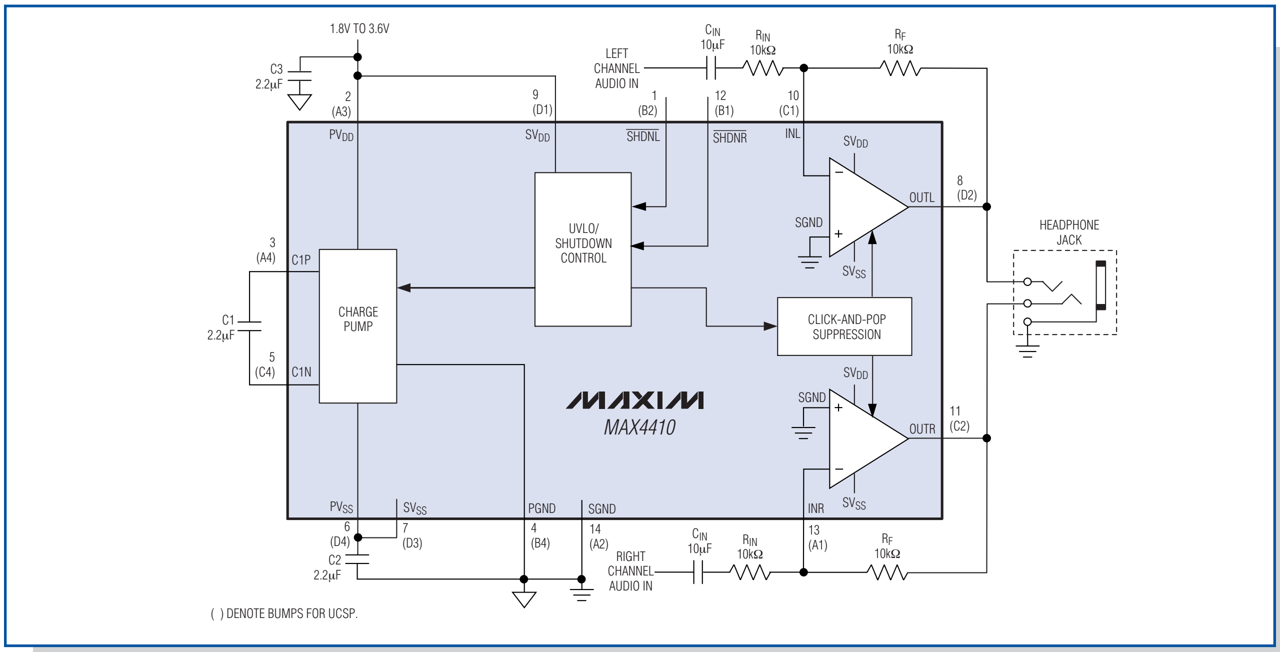


Figure 7. In this typical application circuit for the MAX4410 stereo headphone driver, setting C_{IN} to 10µF restricts any voltage-coefficient effects to subaudible frequencies. Large-valued coupling capacitors at the output are not necessary.

Regarding larger valued capacitors, **Figure 8** compares two types of 100µF capacitors used with a 16Ω resistor in forming a passive highpass filter. At the 100Hz, -3dB frequency, both types contribute significant THD due to the capacitors' voltage coefficient. The 100µF tantalum contribution to THD+N is 0.2% at the -3dB cutoff, which is equal to the worst-performing ceramic device in Figure 4. Eliminating those devices from the audio path using Maxim's DirectDrive components, or similar techniques, improves the audio quality significantly and notably at low frequencies. In Figure 8, a MAX4410 is used to derive the reference plot (limit of measurement).

Summary

Passive components can add real, measurable degradation to an analog audio path. Those effects can be easily examined and assessed using standard audio test equipment. Of the capacitor types tested, aluminum-electrolytic and polyester capacitors give the lowest THD. X5R ceramics give the poorest THD.

When choosing active components, take care to minimize the number of AC-coupling capacitors in analog audio stages. For example, use differential signal paths or DirectDrive components for headphone feeds (e.g., MAX4410). When possible, design audio circuitry with low capacitance values in which C0G or PPS dielectrics can be used. To reduce voltage-coefficient effects in AC-coupled audio stages, restrict potential problems to the subaudio frequencies by lowering the -3dB point much more than necessary, by 10x, for example.

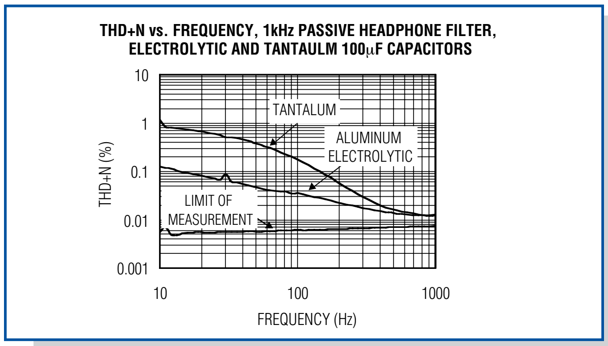


Figure 8. THD+N vs. frequency for large-valued, 100µF capacitors driving a 16Ω load. Both types (tantalum and aluminum electrolytic) contribute heavily to THD at the nominal -3dB point of 100Hz. No such output-coupling capacitors are required with Maxim's DirectDrive headphone amplifiers.