DALLAS SEMICONDUCTOR MAXIM

Microcontroller
ENGINEERING REVIEW

Volume 5

Configuring the MAXQ3120 mixed-signal microcontroller for practical applications

Designers and manufacturers of monitoring applications have repeatedly demanded a microcontroller versatile enough for typical daily applications. The activities could include electricity metering, automotive systems, data gathering, and sensor conditioning. The MAXQ3120 low-power, high-speed microcontroller was designed especially for versatile implementation. Here are the principal specifications garnering attention for this device.

- 16-bit, 8 million instructions per second (MIPS), one-cycle RISC core
- 32kB flash memory
- 512B RAM
- Two UARTs with independent baud-rate generators
- Three timers, one capable of PWM D/A
- Infrared communication capability
- LCD controller capable of driving 112 segments
- Battery-backed real-time clock with time-of-day and subsecond alarms
- 16 x 16-bit one-cycle multiplier with a 40-bit accumulator
- Two precision, 16-bit analog-to-digital converters (ADCs)

With all these impressive features, let us discuss what the MAXQ3120 can accomplish with analog I/O and DSP functions.

The MAXQ3120 can do much more than simply record sound. Coupled with user-interface components and an inexpensive NAND flash memory, the MAX3120 becomes the core of a full-featured voice recorder subsystem.

Voice recorder subsystem

The concept: Give a group of engineers an ADC, and someone will discover how to record voice with it. But the MAXQ3120 can do much more than simply record sound. Coupled with user-interface components and an inexpensive NAND flash memory, the MAX3120 becomes the core of a full-featured voice recorder subsystem.

The details: To perform audio I/O, one of the two ADCs and the PWM timer channel are used. The ADCs have a nominal input of +1V to -1V, and the built-in preamplifier has a programmable gain of up to 16X. In many cases, a condenser microphone cartridge with a built-in impedance-matching amplifier can connect directly to the ADC input. If lower noise or more gain is required, Maxim makes a microphone preamplifier (the MAX4467) that provides the required bias to the microphone and an extremely low-power shutdown mode for battery-operated applications. On the output side, a single-stage amplifier drives the speaker and performs the modest antialiasing and PWM smoothing required.

Table of Contents

At 8MIPS, the MAXQ3120 has sufficient processor horsepower for many of the standard voice codecs in common use. The "gold standard" is the ITU G.711 codec, which operates at 64kbps, transmitting and receiving 8,000 8-bit samples per second.

Once the audio has been converted to a digital sample, it must be compressed and stored for subsequent playback. At 8MIPS, the MAXQ3120 has sufficient processor horsepower for many of the standard voice codecs in common use. The "gold standard" is the ITU G.711 codec, which operates at 64kbps, transmitting and receiving 8,000 8-bit samples per second. The G.711 codec can operate with two different transfer functions that map a 12-bit sample value to an 8-bit codeword; these functions are commonly known as A-law (used primarily in Europe) and μ-law (used primarily in the United States).

If higher compression is desired, the ITU G.726 codec is a possibility, at some sacrifice of voice quality. The G.726 codec uses adaptive delta pulse-code modulation (ADPCM) to more efficiently encode speech without storing the full value of each sample. This codec supports several bit rates down to 16kbps and, for most implementations, requires less than 3MIPS to operate. Both the G.711 and the G.726 codecs require very little RAM.

In the record phase, a timer goes off every 125μs (every 1,000 processor cycles at 8MHz), and the processor averages the samples received during the previous timer period (It is either two or three samples, as the ADC provides a new sample every 48μs.) to match the desired sampling rate of 8kHz. The 16-bit accumulated samples can then be encoded using the selected codec. For playback, the sample data is linearized and then passed to the PWM controller for presentation to the speaker.

Following compression, the audio samples are ready for storage. The MAXQ3120 has no nonvolatile storage other than its program flash, so some form of external storage must be found. The most cost-effective storage solution for this application is NAND flash, which comes in densities up to 8 gigabits. With a 16kbps codec, such a device would provide more than *six days* of voice storage!

NAND flash is not perfect, however. First, most NAND flash devices come with a defect map that tells the application where "dead bits" reside within the memory array. Second, NAND flash cells, like those of all electrically erasable memory devices, tend to lose the ability to change state after extended use. Fortunately, in a voice application both of these types of errors cause fewer problems than they would if the memory were used, say, as a solid-state disk. Consequently, with NAND flash memory, defects of this type may not even be noticed, and if they are, only as a momentary glitch in the audio.

Capable of driving 28 segment lines across four common planes, the MAXQ3120's LCD controller is compatible with a large number of existing 3V LCD glass modules.

With such a vast amount of voice storage, some way of managing the recordings is necessary. This is the job of the user interface, the heart of which is the LCD controller. Capable of driving 28 segment lines across four common planes, the MAXQ3120's LCD controller is compatible with a large number of existing 3V LCD glass modules. Plus, custom LCD modules can be designed and manufactured very cost effectively.

The user controls the recorder through pushbuttons connected to general-purpose I/O ports. There are four 8-bit ports available that are shared with other functions on the device.

What is left to do? The MAXQ3120 is the ideal microcontroller to use in advanced voice-recording systems. There are only a few engineering tasks left to the designer of such a system.

- *Design the user interface:* Select an LCD and decide how information is displayed, which buttons perform what function, and how recordings are organized.
- *Select a voice codec:* You can use one of the ITU codecs mentioned, select another proprietary codec, or store the samples raw if the memory is large enough. Many standard codecs can be purchased as C source code, with only small, low-level interface routines left to be coded.
- *Select a storage medium:* NAND flash is considered the best option, but there are other storage mechanisms that may be more attractive in your application. Removable storage (such as SD, SmartMedia®, or MMC Memory Cards), for example, is inexpensive and ubiquitous, and some manufacturers provide source code in C and development kits to help you design the storage card interfaces.

- *Battery management:* If the recorder is to be used in a battery-operated environment, some form of power management is required. Maxim has many low-cost battery-management solutions. Coupling those solutions with the very low-power stop and sleep modes available in the MAXQ3120 gives the voice recorder an appealing battery life.

Using one of the two ADC channels, the MAXQ3120 can sample the difference signal, extract the strongest frequency component, and scale that to a kilometers-per-hour (or miles-per-hour) reading.

Doppler radar alternative

The concept: Law enforcement has used speed radar for longer than many of us have been driving. If the cost was low enough, this technology could be extended beyond ridding the streets of speeders. For example, Doppler radar could automatically alert the driver if it detected a stopped car ahead. How can the MAXQ3120 help here?

The details: The concept behind Doppler radar is simple. The radar set emits a continuous microwave beam at some known frequency. (In the United States, for example, it is increasingly the Ku band, or about 24.150GHz.) The beam then bounces off a moving object and is reflected back to the radar set, at a frequency slightly higher or lower than the transmitted frequency. The reflected signal is then mixed with the transmitted signal to produce a "beat note" with a frequency defined by:

$$f = [v * (f_0 / c)] * \cos\theta$$

where v is the velocity of the object measured, f_0 is the nominal transmitter frequency, θ is the angle between the object's direction of travel and the radar set (see **Figure 1A**), and c is the speed of light. Notice that if the object is directly approaching the radar set, then θ is zero and $\cos\theta$ is one. The velocity of the target then becomes:

$$v = [f / (f_0 / c)]$$

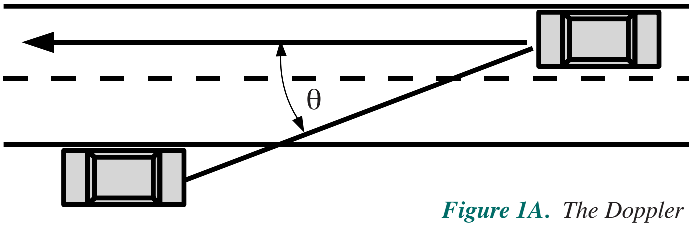


Figure 1A. *The Doppler shift of the received signal is dependent on the target's speed and the angle between the direction of the target's travel and the radar set.*

If, for example, a Ku band radar set produces a 1kHz signal, the target is approaching (or receding) at 12.4 meters per second, or about 28 miles per hour (or 45 kilometers per hour). Processing this audio frequency signal is the MAXQ3120's task. See **Figure 1B** for a block diagram.

Using one of the two ADC channels, the MAXQ3120 can sample the difference signal, extract the strongest frequency component, and scale that to a kilometers-per-hour (or miles-per-hour) reading. Additionally, using the multiply-accumulate unit, sophisticated filtering can isolate the strongest signal and possibly derive intelligence (such as the speed of the operator's own vehicle) from lower-level signals.

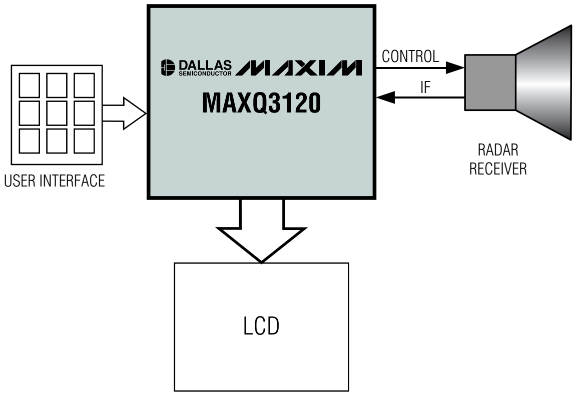


Figure 1B. *In the Doppler radar application, the MAXQ3120 controls a radar head and recovers the "beat note" embedded in the intermediate frequency (IF) signal returned from the receiver.*

In some cases, the user interface is trivial—it may be nothing more than a logic level or contact closure to activate an audible alert. In other applications, the microcontroller may periodically record velocity and the time and date at which the measurement was made.

What is left to do? There are several manufacturers of Doppler radar modules, and most produce an IF in the audio band. For a simple velocity-measuring set, there is little else to do. For a set that performs more complex analysis, some engineering work is required to develop the algorithms for processing the signal. Fortunately, there are excellent tools that aid in the development of complex filtering and discrimination algorithms.

In some applications, it is desirable to know the direction of travel, i.e., is the target approaching or receding? A conventional Doppler radar cannot tell; the frequency shift is the same without regard to the direction of travel. But some manufacturers make a radar module that includes two quadrature outputs. By demodulating both and determining the phase difference between the two channels, the set can determine the direction of travel. As the MAXQ3120 has two ADCs, extending the application for this criterion becomes easy.

Telephone nanny

The concept: You want to track telephone use—who is calling whom, and the time and duration of each call—but you do not want the expense or complexity of a full-featured call-accounting system. For example, parents want to track their children's telephone use. Professionals want to keep an automatic log of whom they call, who calls them, and when. Those who rent rooms by the night in a "bed-and-breakfast" would like to offer telephone use, but find it difficult to track guests' calls when a room telephone is an extension of the house line.

Another MAXQ3120 project holds a solution: a device that monitors all the telephones on a phone line and tracks incoming and outgoing calls (see **Figure 2**).

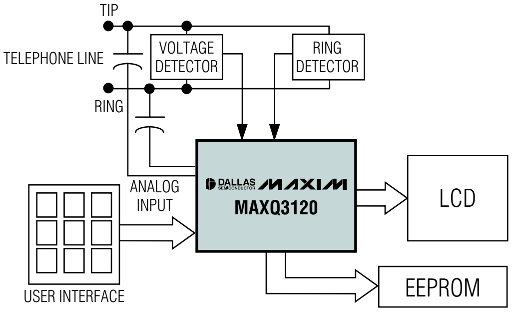


Figure 2. *The telephone nanny monitors dialed digits, incoming ring, caller ID, and off-hook events to determine what numbers are called, what numbers are calling, and how long conversations last.*

The details: The telephone nanny must monitor four types of input: the on-hook/off-hook state of the telephone line, incoming ring signals, dialed numbers, and caller ID signals. The easiest of these tasks is the off-hook signal. Whenever a phone goes off-hook, a voltage detector alerts the MAXQ3120 that the line voltage has changed from about 48V (on-hook) to less than 12V (off-hook).

Incoming ring is caused by a high-voltage AC signal placed on the telephone line by the phone company. To detect this signal, a capacitive-coupled opto-isolator can alert the processor while keeping it isolated from the telephone line. In general, a bidirectional optocoupler in series with 0.47µF and 4.7kΩ will reliably detect an incoming ring. To eliminate false alerts, a pair of back-to-back zener diodes can keep current from flowing in the opto-isolator until the voltage exceeds the breakdown potential.

Receiving dialed numbers is somewhat more complex, as there are two methods used to send digits to the phone company: pulse dialing and tone dialing. In pulse dialing, current is interrupted at 10 pulses per second in a pattern corresponding to each dialed digit. To detect pulse dialing, it is only necessary to time the low-to-high voltage transitions on the line; a series of transitions that occur 10 times per second is undoubtedly pulse dialing.

But fewer and fewer pulse-dial sets are used today. The currently accepted method for sending digits to the phone company is through dual-tone multifrequency (DTMF) signaling. In this scheme, the digits on a telephone are arranged in three columns and four rows, and pressing a key produces one tone corresponding to the row and one tone corresponding to the column on which the key appears. See **Figure 3** for details concerning DTMF frequency assignments. By detecting and decoding the tone combinations, the MAXQ3120 can determine what digits were dialed.

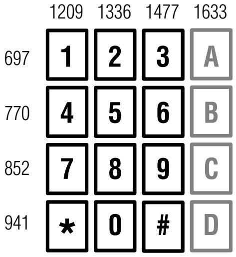


Figure 3. *Dual-tone multifrequency (DTMF) signaling assigns one row tone and one column tone to each key on a keypad. Note that the right column (with the A, B, C, and D keys) is not used in customer applications, and is only present in central office equipment.*

A simple, CPU-efficient mechanism for detecting tones is known as the Goertzel algorithm. This method is implemented as a two-pole filter and provides a clear, unambiguous indication of the tone presence in a potentially noisy channel. MAXQ implementations of the Goertzel algorithm have been coded and tested.

Determining the telephone number of incoming callers is easy with caller-ID service. Subscribers to this service receive a 1200bps signal in the gap between the first and second rings which contains the number that is calling, the name of the caller, and the time and date.

In the United States, caller ID is transmitted from the central office using the Bell 202 modem standard. European standards call out ITU V.32 mode 2 (1300Hz mark and 2100Hz space). In both cases, data transmission is 1200bps. It is easy to build the application to support both standards, but for the purpose of this article we discuss the U.S. standard. In this standard, a "0" bit is represented by a 2200Hz tone, and a "1" bit by a 1200Hz tone. The MAXQ3120 can easily detect zero crossings, and thus determine the frequency of the incoming signal and thereby discriminate the bits. The data format is simple, asynchronous serial in the N81 (no parity, 8 bits with one stop bit) format.

A simple, CPU-efficient mechanism for detecting tones is known as the Goertzel algorithm. This method is implemented as a two-pole filter and provides a clear, unambiguous indication of the tone presence in a potentially noisy channel.

Once the bits have been decoded, the message format must be interpreted. There are two types of caller ID. The first type of message conveys only the calling number, plus the date and time. It looks like this:

TYPE	LENGTH	MONTH		DAY		HOUR		MINUTE		PHONE NUMBER	CKSUM
04	12	30	39	32	33	30	39	35	34	39 30 33 35 35 35 31 32 31 32	49
—	—	09		23		09		54		9035551212	—

This above message type is transmitted if the customer subscribed to "number only" caller ID. Notice that all the characters are transmitted as ASCII except for the type, the length, and the checksum. If the phone number is not available, the letter O is sent instead. If the phone number has been blocked by the customer (or at the customer's request), the phone number field contains the letter P. The checksum is the two's complement of the modulo-256 sum of all the message's previous bytes.

Customers subscribing to "name and number" caller ID receive information that looks like this:

TYPE	LENGTH	DATA BLOCK 0	DATA BLOCK 1	...	DATA BLOCK N	CKSUM

The *TYPE* is always 0x80, and the *LENGTH* is the length of all data blocks. Each data block has the format:

BLOCK TYPE	BLOCK LENGTH	DATA

The *BLOCK TYPE* indicates what kind of data is being transmitted, and is selected from the following values:

VALUE	TYPE
1	Date and time
2	Phone number
4	Number not present
7	Name
8	Name not present

Once the data for a particular call is accumulated, it can be stored in an I^2C* EEPROM. These devices are inexpensive, reliable, and come in a variety of storage capacities. A 16kb EEPROM can store about 100 name-and-number caller-ID entries. An I^2C software implementation is available for the MAXQ family of processors.

What is left to do? There are several enhancements that can be considered. While the project presented here monitors all phones on a line, it cannot tell you *which* phone initiated or answered a call. To do this, a monitoring device would be needed on each phone station, but without the user-interface elements. The MAXQ3120 can use a current sensor to determine when the phone to which it is connected goes off-hook. It then can communicate this fact to the central phone

Incoming ring is caused by a high-voltage AC signal placed on the telephone line by the phone company. To detect this signal, a capacitive-coupled opto-isolator can alert the processor while keeping it isolated from the telephone line.

The MAXQ3120 was built to support an electricity meter application. Its two ADCs are designed to monitor a voltage channel and a current channel.

nanny. To perform this communication task, the station microcontroller sends DTMF digits which identify the station that made or answered the call. In an on-hook state, the phone company would not even "see" the digits, and the phone wiring in the house makes a perfect conduit for these signals.

A second enhancement is automatic logging to a computer. The MAXQ3120 includes UART channels that can connect to the serial port of a PC, essentially turning the phone nanny into a complete call-accounting system in a tiny package. If you combine this project with the voice-recorder project discussed above, you have the core of very sophisticated answering machine or a telephone-call recorder.

Electricity monitoring

The concept: Why is my electric bill so high? This is a common complaint heard by electricity providers. Part of the answer, unrelated to fuel prices, is that we have more and more equipment that stays on all the time.

Do not blame your refrigerator—it actually cycles on and off, switching on only when the interior warms above a preset limit. The real culprits for power usage are all around you. Consider the media equipment with glowing lights that tell you they are off, ready for your remote control to command them on. At one time, an "off" switch meant exactly that, as in *this device is no longer operating in any fashion*. Today, turning a television off simply puts it in standby mode, with much of the circuitry still actively consuming power. In fact, it is difficult to find modern electronic gear that has an honest, circuit-interrupting on-off switch.

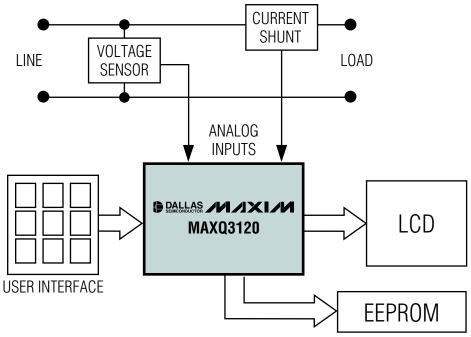


Figure 4. *The power monitor can determine how much power is being used by an appliance and when. It also reports brownouts and voltage surges that can damage sensitive equipment.*

Another covert consumer of power is your personal computer. In the era of always-on Internet, many people leave their PC on to catch downloads, get email, and perform other tasks while they are away. How much does that cost in electricity?

In this final section, the MAXQ3120 returns to its roots, but in the service of the electricity consumer instead of the electricity provider. **Figure 4** shows a block diagram for this application.

The details: The MAXQ3120 was built to support an electricity meter application. Its two ADCs are designed to monitor a voltage channel and a current channel. In this project, the MAXQ3120 continuously monitors the voltage presented to and the current flowing through the device to which it is attached. It then reports the average power used by the device, the times and levels of any usage peaks, and, if needed, the power factor that the device presents to the line.

But how does it report? The simplest and most direct way is by a small LCD on the monitoring device. One or more pushbuttons can command the MAXQ3120 to cycle through its display modes (RMS volts, RMS amperes, watts, watt-hours of energy since last reset, etc.). A small LCD is surprisingly inexpensive and makes a compact, usable package when only one device is monitored.

At just over 20,000 samples per second, the ADCs on the MAXQ3120 cannot demodulate a carrier in the low 100kHz range (the range used by common power-line control systems), but they can demodulate a carrier in the audio range.

If more than one device is to be monitored, having a central station to record the usage from all stations is preferred. The difficulty here is the poor quality of a typical power circuit used as a data-transmission medium. Transmitting data at any significant rate is beyond the ability of the inexpensive modules that we are trying to build.

But transmitting data at what might be considered an *insignificant* rate may be cost effective. At just over 20,000 samples per second, the ADCs on the MAXQ3120 cannot demodulate a carrier in the low 100kHz range (the range used by common power-line control systems), but they can demodulate a carrier in the audio range. And if the data is transmitted slowly enough, about 10bps, communication can be made arbitrarily reliable.

So in addition to monitoring power, the MAXQ3120's DSP functions would perform two additional functions: they would look for power in two narrow passbands to attempt to detect very low-speed frequency-shift keying (FSK) modulation; and when requested, they would generate tones in the 3kHz to 7kHz range for FSK detection by the central-monitoring station.

The central-monitoring station can be a stand-alone unit or a device that attaches to a personal computer through a serial port. The latter is attractive, as the PC has virtually unlimited storage and can perform much more complex reporting functions than any microcontroller.

For power-line communications, both tone generators and bandpass filters have been coded for the MAXQ3120, so building a low-data-rate FSK modem is a matter of plugging together components.

What is left to be done? Not much. A complete power-meter reference design is available at www.maxim-ic.com/MAXQ3120_power that can be easily adapted to this project. For power-line communications, both tone generators and bandpass filters have been coded for the MAXQ3120, so building a low-data-rate FSK modem is a matter of plugging together components. In short, this is a project that can be readily assembled from existing hardware and software components into a working product.

Conclusion

As can be seen from its many application uses, the MAXQ3120 microcontroller is much more than the heart of a multifunction electricity meter. This strong, capable microcontroller offers many opportunities to expand on everyday applications, and may be ideal for your next mixed-signal project.

MAXQ is a trademark of Maxim Integrated Products, Inc.
SmartMedia is a registered trademark of Toshiba Corporation.

*Purchase of I^2C components from Maxim Integrated Products, Inc., or one of its sublicensed Associated Companies, conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

Environmental monitoring with the MAXQ3210

The new MAXQ3210's capabilities make it unique in both the MAXQ family and the embedded microcontroller market. The MAXQ3210 integrates EEPROM code and data storage, a piezoelectric horn driver, and a 9V regulator into a low-pin-count package. A high-performance 16-bit RISC core makes the device both fast and power-friendly. Based on the MAXQ10 core, the MAXQ3210 differs from other MAXQ microcontrollers as it has 8-bit accumulators rather than 16-bit accumulators. The MAXQ3210 can be used in many applications where a few I/O pins and some smart control are required. This article describes some ideal environmental-monitoring applications.

MAXQ3210 features and monitoring capabilities

A high-performance 16-bit RISC core makes the MAXQ3210 both fast and power-friendly.

The MAXQ3210 has 2kB of EEPROM code space, 128 bytes of EEPROM data space, and 64 bytes of RAM. An integrated 9V regulator simplifies the circuit in battery-powered applications. The MAXQ3210 also provides a regulated 5V output for other circuit components. A JTAG debug engine allows in-application debugging without an expensive emulator.

The MAXQ3210 integrates unique peripherals that can be used in environmental-monitoring applications. A piezoelectric horn driver and high-current LED driver provide immediate status feedback when an environmental condition is unsafe or changing. These peripheral capabilities are useful in many monitoring applications; simple security systems, smoke alarms, temperature monitors, and motion detectors all have a place for a microcontroller that drives an electric horn.

Additionally, the device provides multiple options for interfacing to environmental-monitoring circuits. The MAXQ3210's internal analog comparator monitors voltage changes in external circuits, which occur as a result of environmental changes. This external circuit could be something as simple as a thermistor measuring temperature, or something more complex such as a slope analog-to-digital converter (ADC) that measures the amount of time a current takes to charge a capacitor.

Another option for monitoring external circuits is through the MAXQ3210's digital I/O. When an out-of-range condition occurs, for example, the environmental-monitoring circuit generates an external interrupt that awakens the MAXQ3210. The MAXQ3210's I/O pins could also use a serial transmission protocol to communicate data with an external IC that measures distance or lighting conditions.

Software architecture for a monitoring application

Applications written for the MAXQ3210 are generally small and simple enough to be coded in the MAXQ assembly language. For the example application presented later in this article, the MAX-IDE toolset is used. MAX-IDE is a free development environment from Dallas Semiconductor, providing an assembler and a debugging environment for MAXQ devices. **Figure 1** shows the basic architecture for an environmental-monitoring application.

On startup, the device passes through an initialization period in which registers and configuration bits are set for general application use. If the device was just powered on, extra operations may be required, such as manufacturing test and configuration. After passing through initialization and power-on check, the application enters the main loop where it measures and reacts to its environment. First, environmental readings are taken through the comparator or the digital I/O pins, and then analyzed for out-of-range conditions. Next, the application performs periodic diagnostics, which may include testing external circuits, measuring the battery, or checking for permanent faults recorded in the data EEPROM. Following the diagnostics, the application checks the status, which can range from warnings (low battery) to alert conditions (temperature too high). When the environmental readings require action, the application has several options that we discuss below: sound a horn, flash an LED, use the I/O pins to communicate with another device, or simply record the condition into the data EEPROM for later analysis.

Software for a simple monitoring application

A simple application that models an environmental monitor is available online at www.maxim-ic.com/MAXQ3210_Environment. It was built and tested on the MAXQ3210 evaluation kit. A pushbutton toggles between alarming and normal conditions. The horn sounds to indicate an alarm.

The main loop of the environmental-monitoring application appears in the following paragraph. Notice that the state machine for an environmental monitor is very simple; it takes sensor readings and analyzes them to see if the system has exceeded some threshold (temperature too hot, too much smoke in the air, etc.). If the condition is out of bounds, an alarm signals.

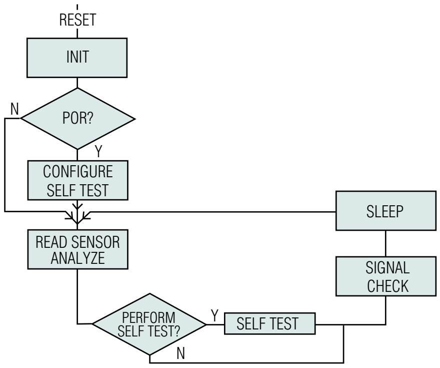


__Figure 1.__ The MAX3210's main program loop in an environmental-sensing application spends most of its time in sleep mode, waking up periodically to read a sensor and analyze the results.

```
MainLoop:
   move  DP[0], #CONDITION_FLAG       ; see if we are alarming
   move  ACC, @DP[0]                  ; read the alarm flag
   jump  z, MainLoop_NoSignal         ; skip next code if not alarming

   ;
   ; If our condition is above threshold, see if it is
   ; time to sound the horn
   ;
   call  CheckSignalTime              ; see if it is time to sound the horn
   jump  nz, ReadAndSleep             ; back to sleep if no signal
   call  SignalCondition              ; sound horn, light LEDs, etc.
   jump  ReadAndSleep                 ; let's go to sleep now
   ;
   ; In a real sensor, we still want to take readings even if we are
   ; signaling.  We need to check to see if environmental conditions
   ; have returned to normal.
   ;
MainLoop_NoSignal:
   call  CheckForSelfTest             ; time to run periodic diagnostics?
   jump  z, ReadAndSleep              ; skip if not time yet
   call  SelfTest                     ; perform self diagnostics

ReadAndSleep:
   call  ReadSensor                   ; get a 'sensor reading'
   call  AnalyzeSensor                ; see if condition out of threshold
   jump  Sleep                        ; put the device into low power mode
```

With its data EEPROM, a 16-bit timer that supports capture, compare, and PWM operations, and high-performance MAXQ microcontroller core, the MAXQ3210 is useful for a wide range of microcontroller applications.

The `SelfTest` function allows periodic system diagnostics, in which applications could monitor their battery condition or check for misbehaving circuits. `SelfTest` is also a good place to increment an internal timer to track how long the MAXQ3210 has been active, thereby allowing the external systems with sensors that degrade over time to have a planned end-of-life.

The application code demonstrates how MAXQ peripherals are easy to use, and how they conserve code space and execution cycles. For instance, the horn driver only requires a single bit to activate or deactivate the horn output.

```
SoundTheHorn:
    move  HORN_DRIVER, #1
    move  LC[0], #10
    call  DelayMilliseconds
    move  HORN_DRIVER, #0
    ret
```

The single-cycle MAXQ core and large register space allow applications to store data efficiently and perform computations quickly. The MAXQ3210 spends more time in low-power sleep mode and less time executing code.

Power management

Power consumption is one of the most important factors in environmental-monitoring applications, which typically run off a battery. The MAXQ3210 provides a low-power stop mode and a low-voltage battery monitor.

When the application is periodically measuring an environmental condition, the MAXQ3210's low-power stop mode has two options for wakeup: an external interrupt or a wakeup timer that can bring the device out of sleep mode and begin code execution. The external interrupt is a good option when the application is waiting for an external circuit to trigger a condition. Typical examples are waiting for a door to open or the voltage across a thermistor to cross a threshold for the external interrupt.

The wakeup timer is another option for bringing the MAXQ3210 out of stop mode. Wakeup is the function discussed earlier in the demo application: the external monitoring circuit wakes up the MAX3210, which measures the environment, reacts if necessary, and then returns to sleep. **Figure 2** shows the typical current-consumption model for such an application. Most of the microcontroller's time is spent in low-power sleep mode. When the device does wake up, the current consumption is much higher. This is where the high performance of the MAXQ core is useful. The MAXQ3210 performs its computations quickly, spending less time in the high-power consumption state and more time in low-power sleep mode.

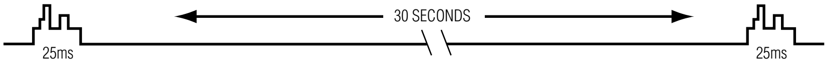


Figure 2. Monitoring applications sleep most of the time to conserve power, waking up periodically for very short runtimes.

Because battery life is a crucial component of most monitoring applications, it is useful to detect when a battery is nearing end of life. The MAXQ3210 determines if battery voltage has fallen below a threshold by simply checking a status bit in a register. This voltage threshold is fixed at 7.7V, which is where 9V batteries begin to break down. At this voltage level, there is ample power left in the battery for the MAXQ3210 to continue to run. A power-conscious application can run for days or weeks on a low battery and issue periodic warning signals, as is commonly done in smoke alarms.

Data EEPROM

The MAXQ3210's 128-byte data EEPROM makes an application smarter. It allows applications to keep permanent configuration and status data, even across power failures or battery removals. Permanent data storage is useful for several purposes.

1) *Yield improvements*. Devices that behave slightly outside specification (for instance, a distance detector that measures a little short) can store permanent configuration information, allowing software to compensate for an external circuit's variations. This allows end devices to be activated or sold that might have previously been discarded.

2) *Behavioral configuration and customization*. MAXQ3210 applications can be customized for their specific target environments or end users. For example, an environmental-monitoring application might be configured as part of a larger network. When the device's measurement is triggered at some threshold, not only would the microcontroller sound its horn, but it could also toggle port pins to alert other devices about the condition. Factory configuration can enable or disable this network notification.

3) *End of life*. In an environmental sensor, the circuit measuring the environment might degrade with use. By updating the EEPROM of the MAX3210 as time passes, an application has control over how long it runs before it must be replaced. A sensor, for example, after five years of running can automatically disable itself, signaling with a horn or flashing LED that it is no longer functional.

Environmental-monitoring applications

Some of the more obvious environmental-monitoring applications for the MAXQ3210 are home-safety applications: fire alarms and gas alarms. The MAXQ3210 has all the tools to implement these applications integrated on-chip. The MAXQ3210 is, however, far more versatile than a dedicated smoke-alarm microcontroller. A variety of applications can be created using the simple environment-monitoring software architecture previously discussed. Some of the following examples target safety applications that prevent or minimize damage to businesses or homes. Other applications provide convenience to the consumer.

The MAXQ3210 has a 128-byte data EEPROM that makes an application smarter. This EEPROM allows applications to keep permanent configuration and status data, even across power failures or battery removals.

To prevent damage to the home or office, one application is a water-level monitor for a basement, where a build-up of water might not be noticed for some time. In this case, water is detected with a humidity sensor or a tank apparatus similar to what is used in a toilet. When the water causes the float to rise above a certain point, the float triggers an external interrupt, and the MAXQ3210 sounds an electric horn to alert residents. In addition, the MAXQ3210 communicates the situation to a larger home or business network, which in turn notifies the business or homeowner about the condition.

Temperature monitoring is another potential application. The contents of a supermarket freezer or a refrigerated car on a delivery truck are monitored for excessive heat. A simple thermistor is used along with the analog comparator; when the temperature of a food cooler exceeds safe limits, the MAXQ3210 indicates the condition to a clerk in the grocery store. This local temperature monitoring has endless useful applications, such as network equipment, beverages, film, laboratory equipment, art supplies, and virtually any perishable product.

Applications can also be about convenience. The MAXQ3210 in a smart motion detector alerts a homeowner when a pet, child, or intruder enters an area of the house that is off-limits. Pushbuttons are used to configure the sensor.

The MAXQ3210 is a natural fit as a parking assistant. Using a simple distance-detection circuit, the MAXQ3210 sounds its horn for different durations depending on the distance measured. This application requires configuration and intelligence in the microcontroller. When placed in the garage, this circuit helps owners park their cars without bumping into the walls. An end user might not want their automated parking assistant sounding the electric horn each time they walk in front of the circuit. Consequently, the device is programmed with an initial delay—when motion is first detected, the system waits two seconds to see if any additional motion is detected. If not, the motion was probably someone walking in front of the sensor. Also, the device could be disabled through the use of pushbuttons; it would be inconvenient if the device constantly beeped while the end user worked in the garage.

Evaluation kit

The MAXQ3210 evaluation kit (EV kit) is an excellent platform to begin prototyping any MAXQ3210 application. It runs off a 9V supply or a 9V battery. Two pushbuttons control the reset and external interrupt signals. A 10-pin JTAG header provides access to hardware debugging routines, thereby allowing viewing and modification of registers, memory, and stack. The I/O pins are connected to a convenient 2 x 20 header, close to a prototyping area for testing external circuits.

An on-board piezoelectric horn and LED can be used to test the sights and sounds of the application. By default, the horn outputs a damped sound—loud, but not painful. Jumpers can be added to the board to short the dampening circuit, allowing the horn to be driven at its full 85dB volume.

The MAXQ3210 EV kit can be used with MAX-IDE. It supports the hardware debug engine of the MAXQ3210, providing source-code-level debugging and memory monitoring.

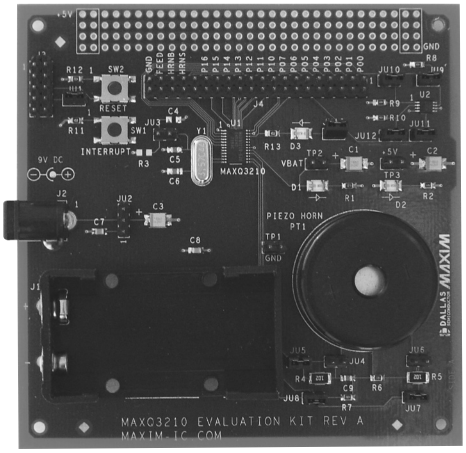


Figure 3. The MAXQ3210 evaluation kit provides a piezoelectric horn, LEDs, and a 9V battery holder for complete application development.

MAXQ3210 benefits summary

As we have seen, the MAXQ3210 has several advantages for environmental-monitoring applications. The primary advantage is integration—components required by monitoring applications (comparator, horn, and LED driver) are integrated onto the chip, eliminating the need for external chips to drive these functions. Integration lowers the overall system cost and improves reliability by reducing the number of components that need to be tested. Plus, a single chip requires fewer connections, lowering the test time for the end circuit board. The single-chip solution also means smaller and less expensive PC boards.

Additional microcontroller benefits are high performance and low-power consumption. The single-cycle MAXQ core and large register space allow applications to store data efficiently and perform computations quickly. The MAXQ3210 spends more time in low-power sleep mode and less time executing code.

Finally, the MAXQ3210's battery monitor and data EEPROM allow smart, self-monitoring applications. Devices can warn the user when their battery is nearing depletion. In addition, applications can track the life of their components and implement a planned end-of-life.

Conclusion

The MAXQ3210 is a low-pin-count implementation of a MAXQ microcontroller, designed for applications that do not require the peripheral support provided by more expensive microcontrollers. While the MAXQ3210 is an excellent fit for environmental sensors, it is truly a general-purpose, high-performance, power-saving microcontroller, capable of adding intelligence and interaction to many applications.

It is important to note that while this article discusses environmental monitoring, the MAXQ3210's applications are far broader. With its data EEPROM, a 16-bit timer that supports capture, compare, and PWM operations, and high-performance MAXQ microcontroller core, the MAXQ3210 is useful for a wide range of microcontroller applications.

Signal filtering with the MAXQ7654

The application presented in this article demonstrates the mixed-signal features of the MAXQ7654. This microcontroller uses the first of its two DACs to output a noisy sinusoid, a low-frequency sinusoid injected with random noise. The DAC output is tied to one of the ADC input channels for voltage measurement. The input samples are run through a simple finite-impulse response (FIR) filter to attenuate the high-frequency components of the signal, resulting in a nice, smooth sinusoid that is output on the second DAC.

With its wealth of analog and digital peripheral support, there are many interesting applications that demonstrate the capabilities of the MAXQ7654. This article focuses on the device's signal-filtering abilities, emphasizing the ADCs, DACs, and the hardware multiply-accumulate unit. Using the IAR compiler and the MAXQ7654 evaluation kit (EV kit), a sample application demonstrates how to filter a noisy sinusoid and output the clean, low-frequency signal underneath.

(**Note:** The source code, project files, and schematics that support this article are available at www.maxim-ic.com/MAXQ7654_Filter.)

The MAXQ7654 includes a hardware multiply-accumulate unit for signal-processing applications. It can multiply two 16-bit numbers in a single cycle, and has an optional accumulate function that operates in signed or unsigned modes.

Integrated analog functions and peripheral components enable signal filtering

The MAXQ7654's 16-channel, 12-bit ADC completes a conversion in as little as 16 clock cycles. At the 8MHz maximum clock rate, it completes 500,000 samples per second. Applications can multiplex up to 16 input pins for single-ended analog measurement, or up to 8 pins for differential signal measurement. The ADC also measures temperature—the MAXQ7654 contains an internal temperature sensor for on-chip (die) temperature readings.

The MAXQ7654 includes a hardware multiply-accumulate unit for signal-processing applications. It can multiply two 16-bit numbers in a single cycle, and has an optional accumulate function that operates in signed or unsigned modes. This facilitates the implementation of FIR and IIR filters; each coefficient of the filter requires as little as three machine cycles to process, plus some overhead each time the filter is invoked.

A JTAG debug engine, which is common to the MAXQ platform, provides read and write access to registers and memory while applications are running on the real hardware. JTAG also eliminates the need for expensive emulators. Major C compiler vendors such as Rowley, IAR, and Python support the MAXQ7654 and its debugging capabilities.

A new peripheral for the MAXQ platform is a controller area network (CAN) 2.0B interface, a common network protocol in industrial and automotive applications. Capable of bit rates up to 1Mb per second, the MAXQ7654's CAN controller supports 15 message centers. Interrupts notify the system when messages are received or sent.

An SPI™ interface supports slave or master mode and 8- or 16-bit data transfers. SPI is commonly found in small integrated circuits such as programmable battery chargers, digital potentiometers, DACs, ADCs, and memory chips.

The MAXQ7654 has four multipurpose timers. These timers are configurable for 8- or 16-bit counting, and support auto-reload for periodic interrupts, pulse-width modulation, capture, and compare functions.

Software architecture for the filtering application

The noisy sinusoid is output on the first DAC in a timer interrupt to ensure that output samples are transmitted at regular intervals. However, the code to generate a sinusoid involves complex floating-point calculations, and is computationally expensive. Plus, a sinusoid is periodic, repetitive data. Recalculating sinusoid data that will not change over time is a waste of resources. Therefore, upon startup, the application precomputes an array of sinusoid data.

Based on the high-performance, 16-bit RISC MAXQ20 core, the MAXQ7654 offers a 16-channel, 12-bit analog-to-digital converter, and dual, 12-bit digital-to-analog converters.

After the sinusoid data is initialized, the timer is configured to generate periodic interrupts. In the timer interrupt code, a pseudorandom number generator computes noise, which is simply added to the clean sinusoid value. The result is passed to the DAC for output conversion.

To keep the demonstration code simple, the analog input signal is sampled in the same timer interrupt used to output the noisy signal. When an input sample is read, it is run through a simple FIR filter, which is implemented in assembly language for maximum efficiency. The filtered sample is then output on the second DAC. An oscilloscope is used to compare the two DAC outputs. One sinusoid is jagged and noisy, while the other sinusoid appears clean, with a slight phase delay due to the length of the FIR filter.

Generating and sampling a noisy sinusoid

The timer interrupt code shown below starts with a precomputed sinusoid value and converts it to a noisy sinusoid value.

```
sample =  static_sin_data[sinindex++];
sinnoise = ((sinnoise ^ 0x5C) * 31) + 0xabcd;
thisnoise = sinnoise;
if (thisnoise & 0x01)
{
     thisnoise = thisnoise & 0x1ff;
}
else
{
     thisnoise = -1 * (thisnoise & 0x1ff);
}
sample += thisnoise;
if (sample < 0)
     sample = sample * -1;
if (sample > 4095)
     sample = 8192 - sample;

DACI1 = sample;          // Send value to DAC #1
if (sinindex >= SIN_WAVE_STEPS)
     sinindex = 0;
```

The MAXQ's hardware multiply-accumulate unit is easy to use—filter coefficients and input samples are loaded into the multiplier registers, and the multiplication result is ready after one clock cycle.

The variable `sinnoise` stores pseudorandom noise, which can be positive or negative. The noise factor is added to the value of the pure sinusoid, and the resulting noisy sinusoid value is simply assigned to the DACI1 register for digital-to-analog conversion.

Reading a sample from the DAC is nearly as simple. After selecting the input pin for the ADC to sample, software can either poll a busy bit or enable an interrupt to be notified that the conversion is complete. The sample code uses the polling technique.

```
inputsample = ADC_Convert_Poll(AIN0 | START_CONV | CONTINUOUS);
...
unsigned int ADC_Convert_Poll (unsigned int Control_Reg)
{
  ACNT = Control_Reg;                // Set the ADC parameters
  while( ACNT_bit.ADCBY == 1);       // Wait till ADC is not busy
       return ADCD;                  // Return the ADC result
}
```

Remember that the sampling rate for the ADCs on the MAXQ7654 is 500ksps. With an 8MHz clock, the code spends only 16 clock cycles waiting for a conversion to complete.

Designing a simple digital filter

The Java applet outputs the actual transform realized by the fixed-point coefficients and a graphical representation of the error.

The noisy waveform generated in this application contains one strong low-frequency signal and a large amount of high-frequency noise. A simple lowpass filter cleans this signal.

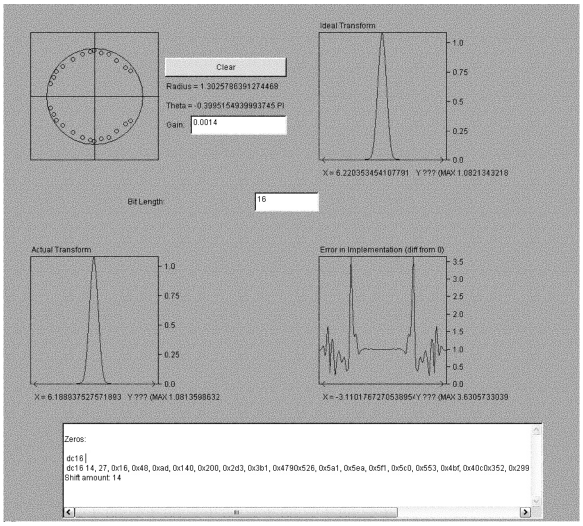
A general FIR filter is an equation of the form:

$$Y = \Sigma \quad A_n * X_n$$

where A_n represents the filter coefficients, X_n is the previously sampled inputs, and Y is the current output of the filter. The filter coefficients determine the frequency response of the filter, or how the different frequency components are attenuated or accentuated.

A Java applet (available in the source code distribution for this article) was used to generate filter coefficients based on a pole-zero plot (**Figure 1**). The applet produces a set of high-precision floating-point filter coefficients. However, because the MAXQ7654 has a 16-bit hardware-multiply accelerator, the floating-point coefficients need to be converted to fixed-point coefficients with 16-bit precision. This conversion introduces error to the ideal filter transform. Therefore, the Java applet also outputs the actual transform realized by the fixed-point coefficients and a graphical representation of the error. Note that while the applet supports both poles (which accentuate frequency components) and zeros (which attenuate frequency components), the demonstration code only uses zeros. Infinite-impulse response filters (containing both poles and zeroes) can be implemented with additional software support.

The text box at the bottom of the applet produces the 16-bit fixed-point filter coefficients, plus the number of decimal places in the fixed-point numbers.



Radius = 1.3025786391274468

Theta = -0.3995154939993745 PI

Gain: 0.0014

X = 6.220353454107791 Y ??? (MAX 1.0821343218

Bit Length: 16

X = 6.188937527571893 Y ??? (MAX 1.0813598632

X = -3.1101767270538954Y ??? (MAX 3.6305733039

```
Zeros:

 dc16 |
 dc16 14, 27, 0x16, 0x48, 0xad, 0x140, 0x200, 0x2d3, 0x3b1, 0x4790x526, 0x5a1, 0x5ea, 0x5f1, 0x5c0, 0x553, 0x4bf, 0x40c0x352, 0x299
Shift amount: 14
```

Figure 1. *This illustration shows the output of the Java applet that produces the filter coefficients. The applet produces the ideal transform, actual transform, error, and 16-bit filter coefficients.*

Implementing an efficient digital filter

This section discusses how the fixed-point coefficients are implemented in a real digital filter. The digital filter algorithm is coded in assembly for maximum performance. This allows application developers to optimize the filter routine based on the requirements of an individual application. Squeezing in an extra cycle or two can make a significant impact on the maximum filter length and sample rate that an application can support.

This demonstration makes two key decisions to maximize the filter's efficiency. First, this application uses an unrolled filter loop. This increases the code size of the algorithm, but produces a very fast filter, requiring three cycles and three codewords per coefficient. This design decision is not unrealistic. A high-quality filter designed with a Kaiser window might produce a filter with 250 coefficients, yielding a total code-size cost of 750 words. On a machine with 65,536 words of code space, this is a valid decision if filter performance is important.

The digital filter algorithm is coded in assembly for maximum performance.

The second key decision to improve filter efficiency is to dedicate 256 words of RAM to a circular buffer that stores the previous input data (the Xn portion of the general filter equation). If the filter has 250 coefficients, the application must store 250 previous input values anyway, so dedicating 256 words of RAM to the filter is not wasteful. The benefit of this decision is that the MAXQ's base-offset pointer can be used to create a circular buffer in hardware. The filter algorithm does not need to check if a pointer has reached the start of a data buffer, because the pointer automatically rolls over the buffer boundary. The following is the code for the digital filter.

```
filtersample:
    push   DP[1]                       ; preserve IAR's software stack
    push   DPC                         ; probably needs this preserved
    move   AP, #0                      ; select accumulator 0
    sub    #2048                       ; normalize the input sample
    move   DPC, #10h                   ; DP[0] byte mode, BP word mode
    move   BP, #W:sampletable          ; start of the sample table
    move   DP[0], #B:sampleindex       ; point to sample current index
    move   AP, #1                      ; select accumulator 1
    move   ACC, @DP[0]                 ; get current table index
    move   Offs, ACC                   ; put it in the offset register
    add    #1                          ; increment the current index
    move   @DP[0], ACC                 ; restore the table pointer
    move   @BP[Offs], A[0]             ; store the current sample
    move   MCNT, #22h                  ; signed, accum, clear regs first

filterloop:
    ;
    ;      Unroll the filter loop for speed.
    ;
    move  MA, #0x16
    move  MB, @BP[Offs--]
    move  MA, #0x48
    move  MB, @BP[Offs--]
    ...
    move  MA, #0x7
    move  MB, @BP[Offs--]
    move  MA, #0x2
    move  MB, @BP[Offs--]
    nop

    move  A[2], MC2              ; get MAC result HIGH
    move  A[1], MC1              ; get MAC result MID
    move  A[0], MC0              ; get MAC result LOW
```

The MAXQ7654's analog-to-digital converter completes a conversion in as little as 16 clock cycles. At the maximum 8MHz clock rate, it completes 500,000 samples per second.

The code first normalizes the input sample. Because the MAXQ7654 has a 12-bit ADC, the input values range from 0 to 4095. To use the digital filter, the input values should be normalized to -2048 to +2047, so subtraction by 2048 is performed ($2048 = 2^{11}$). Once the pointer to the input samples is initialized and the current input sample is stored, the code executes the filter.

The MAXQ's hardware multiply-accumulate unit is easy to use. Filter coefficients and input samples are loaded into the multiplier registers, and the multiplication result is ready after one clock cycle. The input samples are read from the BP[Offs] pointer, and the filter coefficients are hard-coded, taken directly from the output window in Figure 1 (reproduced here):

```
dc16 14, 27, 0x16, 0x48, 0xad, 0x140, 0x200, 0x2d3, 0x3b1, 0x479, 0x526, 0x5a1,
0x5ea, 0x5f1, 0x5c0, 0x553, 0x4bf, 0x40c, 0x352, 0x299, 0x1f4, 0x163, 0xf0,
0x97, 0x58, 0x2e, 0x15, 0x7, 0x2
```

The "14" in the first line means that the numbers in the filter have 14 places after the radix point, and the result must be shifted 14 places to the right when the filter is complete. The "27" means that there are 27 coefficients in the filter. Following those control values, the coefficients are listed starting with A_0 (0x16, 0x48, 0xad, . . .).

After the filter algorithm is complete, the accumulated result is ready in the multiply-accumulate unit's registers MC0, MC1, and MC2. The result must be shifted to compensate for the fixed-point radix.

To change filters used by the application, simply alter the code underneath the filterloop label. For each coefficient output by the Java applet, add the instruction pair:

```
move   MA, #COEFFICIENT_n
move   MB, @BP[Offs--]
```

Also, make sure to change the shift count if necessary.

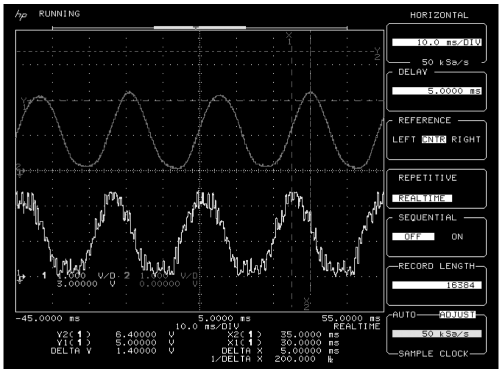


Figure 2. The bottom signal is the noisy DAC output from the MAXQ7654. It is sampled, filtered, and output as the top signal.

Results

The simple filter does its job perfectly. **Figure 2** shows an oscilloscope capture of the two MAXQ7654 DACs. Notice the phase shift on the clean output signal due to the length of the FIR filter.

Evaluation kit

Schematics for the MAXQ7654 EV kit are available with the source code distribution for this application. The kit board has many options for exploring the MAXQ7654 microcontroller. Jumpers select supply voltages and peripheral configurations, and every pin is accessible on the board. The MAXQ7654 EV kit (see **Figure 3**) also integrates the JTAG hardware, so no external board is required for loading or debugging.

Conclusion

As we have seen, the MAXQ7654 is a high-performance, mixed-signal microcontroller with a wide range of applications. With the MAXQ7654's simple demonstration code and integrated designs for maximum performance, the device offers designers easy-to-use elements for their signal-filtering application.

SPI is a trademark of Motorola, Inc.

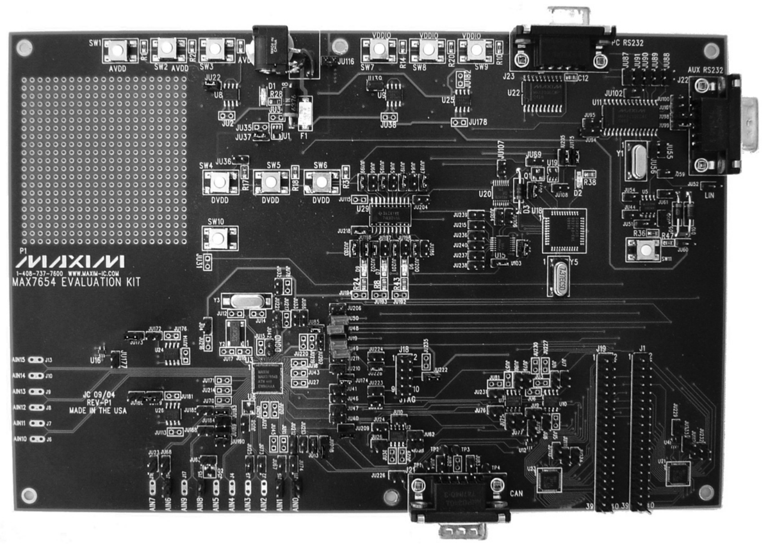


Figure 3. With plenty of I/O, pushbuttons, and a prototyping area, the MAXQ7654 EV kit is an ideal platform for evaluating the MAXQ7654.

Security system control with the MAXQ2000

Common alarm-control panels contain several input devices and require user displays. The usual components for these systems include:

- A device to accept input from the user: a 4 x 4 switch keypad.
- A device to display output to the user: an LCD display.
- An input device: a magnetic reed switch.
- An output device: a piezoelectric horn.

A small piezoelectric horn can be interfaced with the MAXQ2000 by connecting it between two port pins.

These several components can be managed and controlled by a simple application and the powerful, flexible MAXQ2000 microcontroller. This application, available online at www.maxim-ic.com/MAXQ2000_Alarm, was written in MAXQ assembly language using the MAX-IDE development environment. The code was targeted for the MAXQ2000 evaluation kit board, using the following additional hardware:

- Keypad: Grayhill 16-button (4 rows by 4 columns) keypad 96BB2-006-F
- Piezoelectric horn: CEP-1172
- Magnetic reed switch: standard single-loop type

Design goals

Our example application performs the following tasks:

- Monitors the magnetic reed switch to determine if a door/window is open or closed.
- Allows the user to arm or disarm the system by entering a PIN on the keypad.
- Displays status information to the user on the LCD.
- Provides audio indications of keypresses and sensor open/close events by sounding the piezoelectric horn.
- Sounds the horn continuously if the sensor is opened while the system is armed.

The behavior of the alarm control application consists of four discrete states: CLOSED, OPEN, SET, and ALERT (**Figure 1**).

Interfacing to the magnetic reed switch

In an alarm system, magnetic reed switches are installed in two parts: a magnet and the actual reed switch.

In an alarm system, magnetic reed switches are installed in two parts: a magnet and the actual reed switch. The magnet portion is placed on the moving section of a door or window, while the switch portion is placed on the frame. When the door or window is closed, the magnet closes the reed switch, indicating a nonalarming condition. If the system is armed and the window or door is opened, the reed switch changes state, allowing the MAXQ2000 to sound an intrusion alert.

The reed switch is interfaced to the MAXQ2000 simply by connecting it between port pins P5.2 and P5.3. With P5.2 set to an active-low pulldown (PD = 1, PO = 0) and P5.3 set to a weak pullup input (PD = 0, PO = 1), P5.3 will read zero when the reed switch is closed and one when the reed switch is open.

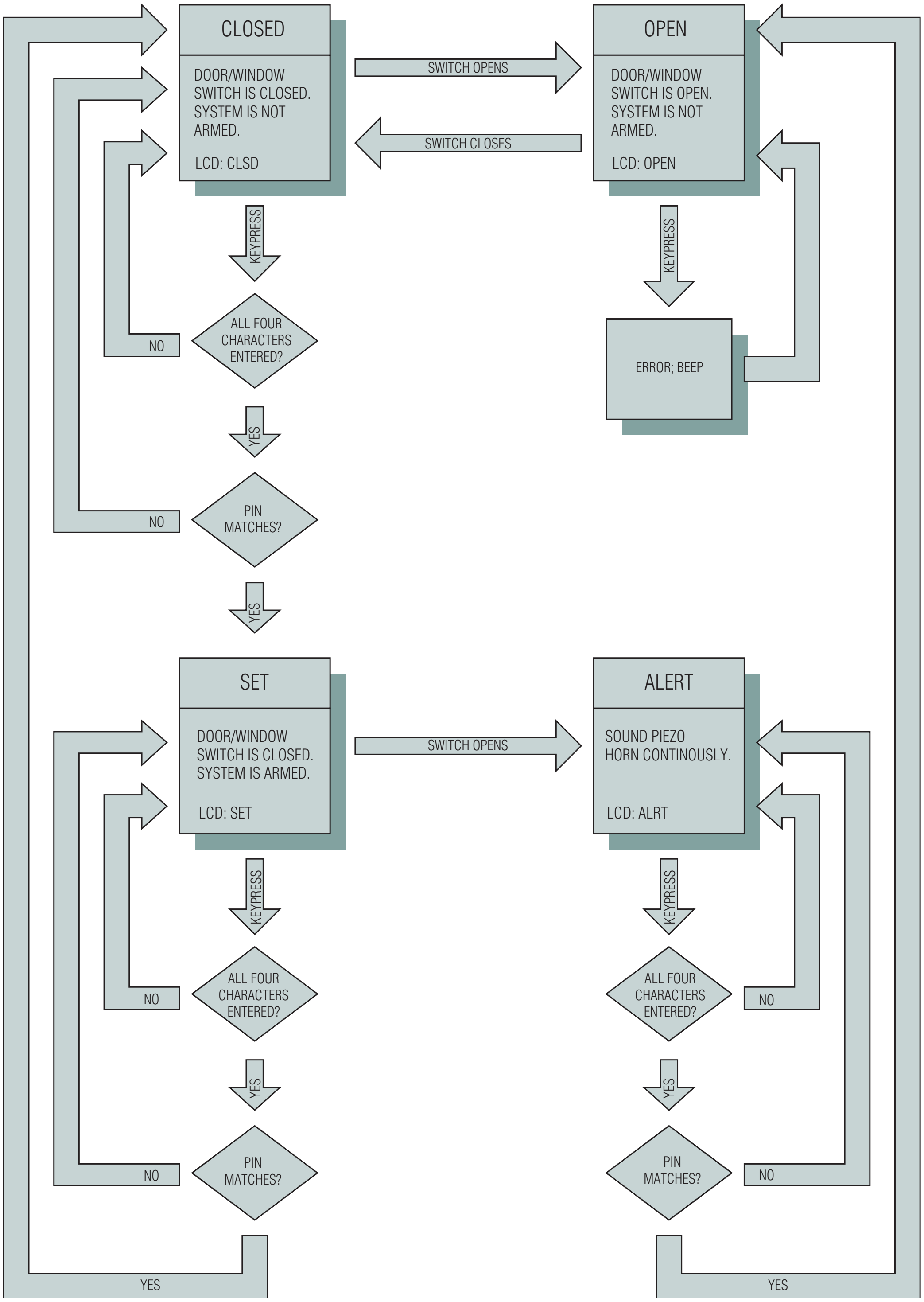


Figure 1*. The alarm control application operates in four main states: CLOSED, OPEN, SET, AND ALERT.*

Interfacing to the 4 x 4 keypad

Keypads are used in alarm control systems for secure PIN entry, to arm/disarm the system, and to change configurations. The keypad used in this example application consists of 16 switches, organized in a 4 x 4 grid. The switches are tied together in a row and column matrix (**Figure 2**) so that depressing a keypad switch connects one row line to one column line. For example, depressing the "3" key connects row 1 and column 3 together.

The keypad provides eight interface pins, one pin for each row and column of the keypad matrix. The keypad and the MAXQ2000 EV kit are connected as shown.

Pin	1	2	3	4	5	6	7	8
Connect	Row 1	Row 2	Row 3	Row 4	Col 1	Col 2	Col 3	Col 4
Port Pin	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P6.6	P6.7
JU2 Pin	54	52	50	48	46	44	42	40

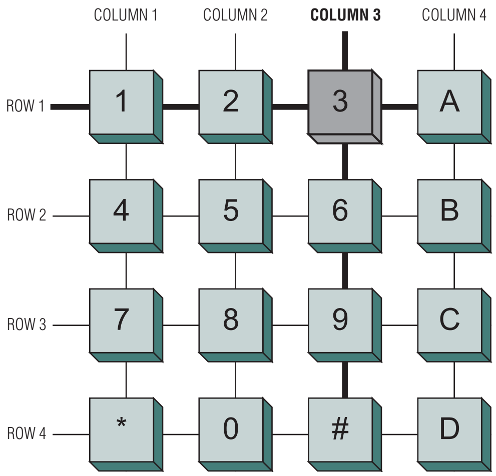


Figure 2. *The keypad switches form a grid of four rows and four columns.*

For this application, the EV kit board should be configured as described below.

- DIP switches
  - The following switches must be OFF: all SW1 switches, SW3.1, SW3.7, SW3.8, SW6.1, SW6.4, SW6.5, SW6.6, SW6.7, and SW6.8.
  - All other DIP switches can be in any state.
- Jumpers
  - The following jumpers must be OPEN: JU5, JU6, JU8, and JU9.
  - The following jumpers must be CLOSED: JU1, JU2, JU3 and JU11.
  - All other jumpers can be in any state.

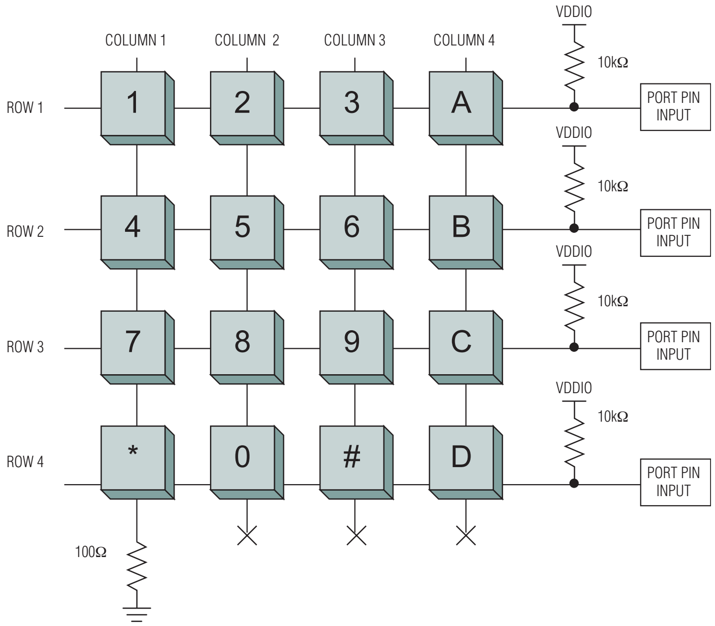


Figure 3. *The MAXQ2000 pulls column 1 low to read the state of the first four keypad switches.*

Scanning by columns

The row and column arrangement of the keypad makes it easy to read the state of four switches at any one time, on either a row or column basis. To read four switches in one column, first the line for that column must be pulled low, and all other columns tri-stated (**Figure 3**). Next, a weak pullup must be set on each row line. Finally, the four row lines are connected to port pin inputs. The input from a row will be low when the switch on that row is depressed, and high otherwise.

Similarly, the state of four switches in a row can be read by pulling that row line low and setting inputs and weak pullups on all four columns. The rows and columns are interchangeable.

In our setup, the four row lines (keypad pins 1 through 4) are all connected to the same input port (P6[3:0]), which makes it easier to read them simultaneously. For this reason, the example application scans one column of switches at a time.

There are four setup states (see table below) for the eight port-pin lines connected to the keypad, each of which allows four of the switches to be read. All input lines read low when the switch being read is closed, and high when the switch is open.

With the row and column arrangement of the keypad, it is possible to read the state of four switches at any one time.

STATE	P6.0	P6.1	P6.2	P6.3	P6.4	P6.5	P7.0	P7.1
1	Input - 1	Input - 4	Input - 7	Input - *	low	tri-state	tri-state	tri-state
2	Input - 2	Input - 5	Input - 8	Input - 0	tri-state	low	tri-state	tri-state
3	Input - 3	Input - 6	Input - 9	Input - #	tri-state	tri-state	low	tri-state
4	Input - A	Input - B	Input - C	Input - D	tri-state	tri-state	tri-state	low

An interrupt-driven state machine

The four columns must be strobed quickly so that any keypress has time to be read before it is released. Additionally, to prevent a switch's bouncing contacts from registering multiple presses, a key must be held down for a certain amount of time before it registers. Both of these factors can be done at once by making a timer-driven interrupt routine the heart of the application. This allows the application to scan through each one of the four columns in a periodic manner and to count the length of time a key has been depressed.

The reload value for the timer controls how often the interrupt will fire. This value must be short enough so that all keypresses are recognized. Additionally, to ensure that key response is not sluggish, the reload value must also be long enough so that it does not occupy an excessive amount of processing time. The value 0FF00h used in the example application code (once about every 2.4ms) was reached through experimentation.

Once the column line for a group of four switches is driven low, some time may be required for the connection operating through a depressed switch to pull its input line low. This time is affected by the switch's on-resistance and by how many column switches are depressed at once. To avoid having to delay the interrupt service routine between pulling the column line low and reading the four switches, the column line for a given state is driven low in the previous state (**Figure 4**).

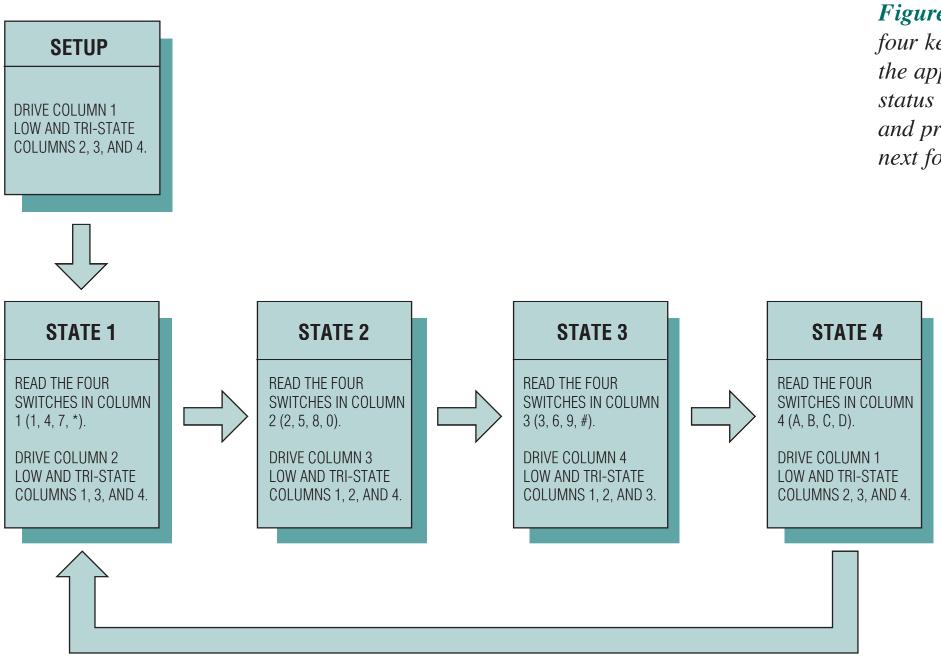


Figure 4. *In each of the four key-scanning states, the application reads the status of four switches and prepares to read the next four.*

In the CLOSED, SET, and ALERT states, a PIN can be entered to change the alarm controller to another state.

Because the interrupt vector (IV) for the MAXQ2000 can be set on-the-fly, the application holds the next-state value in the interrupt vector register. Whenever the timer interrupt fires, the handler routine for the current key-scanning state sets the interrupt vector address to the next state's handler routine.

The handler routines for the other four states are similar, with a slight adjustment to OR in the previously collected switch bits in the A[13] holding register. There are three working accumulators used by the state routines.

A[13] holds the bit array of all the switch states read on the current pass through the keypad. After the State 4 read completes, this register contains the following bits, where a one bit represents an open (released) key switch and a zero bit represents a closed (depressed) key switch.

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
*	7	4	1	2	5	8	0	3	6	9	#	D	C	B	A

Debouncing switches

After State 4 is reached and all keys are scanned, a decision must be made whether to accept any keys that are pressed. A simple way to handle debouncing is to maintain a counter value for each of the 16 switches. Every time State 4 is reached and the key is pressed, the counter is incremented. If the key is not pressed, the counter is decremented. When the counter reaches a certain value, the keypress is registered. To prevent a held-down key from repeating (which typically is allowed on computer keyboards, but not on keypads), the counter must be allowed to decrement back to zero (by releasing the key) before that key may be registered again.

As we have the state of all 16 keys in a single register, there is a simpler, less memory-intensive solution for debouncing. The application maintains a single counter value that is incremented each time the bit pattern matches the pattern read on the previous pass.

To prevent keys from repeating, once a bit pattern has been static long enough to be accepted, a different bit pattern (which includes the idle state where no keys are depressed) must be accepted before the first bit pattern can be accepted again.

Handling simultaneous keypresses

Simultaneous keypresses are possible when using a keypad input device. The debouncing code ensures that if a second key is pressed right after the first, the debounce interval will start over, but be short enough in practice so that this is not an issue.

Once a bit pattern has been accepted, the action for each depressed-key bit can be taken by rotating all 16 bits into the carry bit individually using the accumulator and checking each in turn. The example application code responds only to the first depressed key, but this could be easily changed.

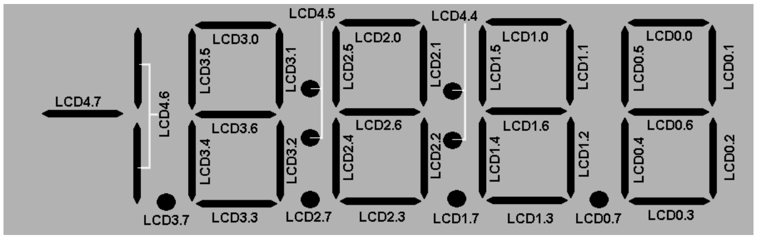


Figure 5. *The LCD display contains four-and-a-half 7-segment characters.*

Interfacing to the LCD display

The LCD display included with the MAXQ2000 EV kit has segments defined as shown (**Figure 5**).

First, the LCD display must be initialized to static drive mode and enabled. Once this has been done, characters can be written to the display by setting segments appropriately.

The MAXQ2000 interfaces easily and directly to LCD displays by means of its dedicated LCD controller peripheral.

Entering the PIN

In the CLOSED, SET, and ALERT states, a PIN can be entered to change the alarm controller to another state. As each character is entered, the working value held in A[10] is shifted left and ORed with the new character, and the decimal point on the LCD display moves left to indicate the number of characters entered. For security reasons, the PIN being entered is not shown on the display.

Once all four characters are entered, the PIN is checked against a hard-coded value. If the entered value matches the PIN, the appropriate state transition occurs.

Using the piezoelectric horn

In our application, a small piezoelectric horn is used to perform two functions: (1) provide audio feedback when keys are pressed or when an incorrect PIN is entered, and (2) sound an alarm when the reed switch opens while the system is armed.

For demonstration purposes, a small piezoelectric horn can be interfaced with the MAXQ2000 by connecting it between two port pins. The port pins are driven differentially to increase the current drive to the piezoelectric horn, and the loop counts used in the driver code determine the frequency of the tone emitted.

In an actual alarm system, stronger drive circuitry would be used to run the piezoelectric horn, and the horn would be driven at its resonant frequency to increase the volume.

Conclusion

The MAXQ2000 interfaces easily and directly to LCD displays by means of its dedicated LCD controller peripheral. Multiplexed keypads can be read in a straightforward manner using the flexible port-pin configuration provided by the MAXQ2000. A timer-interrupt-driven state machine allows all keys in the matrix to be scanned and debounced with minimal effect on processor overhead. Finally, a piezoelectric horn and magnetic reed switch can be controlled easily as well, using the general-purpose port pins available on the MAXQ2000.