CIRCUIT CELLAR INK®
THE COMPUTER APPLICATIONS JOURNAL

FEATURE ARTICLE

Brad Stewart

Low-Cost Voice Recognition

Brad's Tiny Voice—based on an 'HC705 and powered off a 9-V battery—can be trained to recognize up to 16 command templates and costs less than $5. Toys, voice-activated padlocks, and remote controls had better listen up.

Voice recognition has come a long way in the past five years, due mainly to the advent of cheap and powerful PCs equipped with Pentiums and MMX technology. Performance continues to improve to the point where parts of this article were comfortably voice-dictated via Kurzweil VoicePlus.

But, this performance comes at a cost. You need fast Pentiums with MMX, at least 16 MB of DRAM, and even more disk stroage.

What if your application has a budget of a couple dollars? Can you still embed some form of voice recognition or voice command and control into your product?

In this article, I'll show you how to implement a voice-command system for under $5. I conclude with some application examples and recommendations to improve the system even further.

TINY VOICE

My system—Tiny Voice—is based on a low-cost, 20-pin single-chip controller. It's a speaker-dependent, template-based, isolated-word recognizer. You train it to recognize your voice.

Up to 16 voice patterns are stored in a nonvolatile 512-byte serial EEPROM. Five push buttons enable programming and operation, and seven LEDs give status.

For embedded systems, Tiny Voice can be controlled over a parallel or serial protocol from a host microcontroller or it can run stand-alone. The source code may be modified to fit your requirements.

At under $5, Tiny Voice won't do dictation. But, it's good for applications like toys, repertory phone dialers, voice-activated padlocks, security systems, remote controls, and other low-cost consumer products.

A voice command can be one or several words, with a total maximum length of 1.6 s and a minimum of 0.2 s. Response time is typically <100 ms. By carefully selecting the vocabulary and context, over 95% recognition accuracy is possible.

The heart of the system is the 68HC-705J1A Motorola 8-bit processor. There were a number of reasons why I chose this part over a comparable one from Zilog or Microchip.

There's sufficient RAM (64 bytes) to buffer the input waveforms and hold template structures, and its 1240 bytes of ROM provide enough program storage. Also, interrupts are supported, including changes on the I/O lines.

This system is inexpensive (<$2) in high volume. The development kit is cheap, too, at $99.

Shown in Photo 1, the Tiny Voice system was built on a 3″ × 3″ breadboard and is powered off a 9-V battery. Standby current consumption is ~2 mA, which is primarily due to the op-amp and electret microphone bias.

With some added power management, standby current could be reduced to a few microamps. Operating power while sampling and analyzing speech is ~10 mA.

THEORY OF OPERATION

The 68HC05 processor is very simple. There are no ADCs, so you need a way to convert the time domain signal to a format the microcontroller can recognize.

The small amount of memory requires a lot of approximations and simplifications to convert the speech into a small set of features.

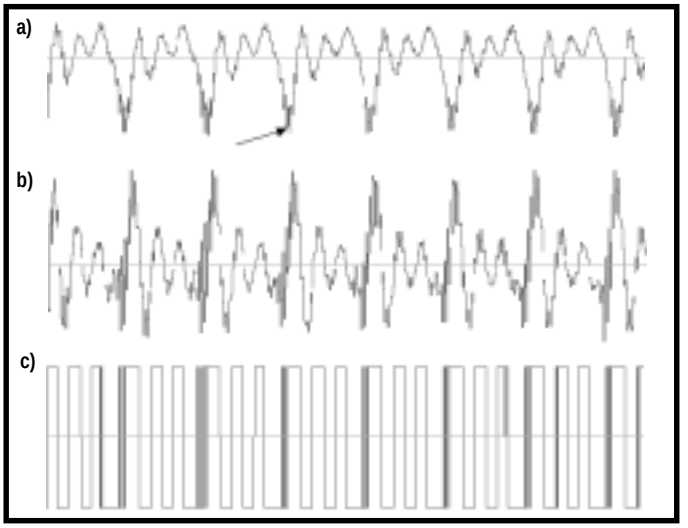

Figure 1a—*This is a waveform of the voiced sound "ee" as in "speech." The arrow points to high-frequency wiggles corresponding to the second formant (F2). Note that these wiggles do not cross the zero axis.* ***b—****After preemphasis or high-pass filtering, the F2 components now cross the zero axis with the same waveform.* ***c—****After being infinitely clipped, the waveform of Figure 1b is a square wave showing both F1 and F2 components. This signal is applied to the microprocessor via a digital input pin.*

To meet these limitations, I use a simplified formant tracker. The microphone input is high-pass filtered and then infinitely clipped using two operational amplifiers. The resulting square wave is connected to an MCU input.

By sorting and tallying long and short pulse widths of the square wave, you get a crude but effective two-channel frequency analyzer. One channel gives frequencies below 1500 Hz, and the other ranges from 1500 Hz to 5 kHz.

These two frequency areas roughly define F1 and F2, the two formant regions of speech. It's a well-known principle that F1 and F2 for a given speaker and a given set of vowels remain the same.

Using F1 and F2 was first tried in 1952 by Bell Labs employing vacuum tubes and capacitors for memory. Crude as it sounds, that system achieved 97% recognition accuracy!

The input signal is high-pass filtered (i.e., pre-emphasized) to accentuate the F2 frequencies. Figure 1 illustrates why this is necessary.

Figure 1a is a sample of the voiced vowel sound "ee" as in "speech." Note the F2 component shown by the arrow. Also note that these high-frequency wiggles do not cross the zero axis. Thus, if the waveform is infinitely amplified and clipped, the square wave would not reveal the F2 component.

However, Figure 1b shows what happens after pre-emphasis. The F2 wiggles cross the zero axis, and the resultant infinitely clipped square wave now contains both F1 and F2 (see Figure 1c).

TINY HARDWARE

Figure 2 shows a schematic of the system. An electret condenser microphone is biased to 5 V via R4. The signal is then amplified by U2a.

C2 and R6 (along with C3 and R10) form a high-pass filter, with a cut-off frequency of 1600 Hz with an added zero at 800 Hz. This setup provides a pre-emphasis function.

C1 serves as a mild antialiasing low-pass filter. The output is fed to the second op-amp, which is configured as a comparator with some hysteresis. R8 sets the threshold of the comparator.

The comparator's output is a square wave that's applied to an input pin of the processor. The threshold defines the beginning and end of a speech utterance. With no signal present, the second op-amp's output is at a DC level.

Voice pattern data is stored in a nonvolatile EEPROM. For this project, I selected Ramtron's FM24C04, which

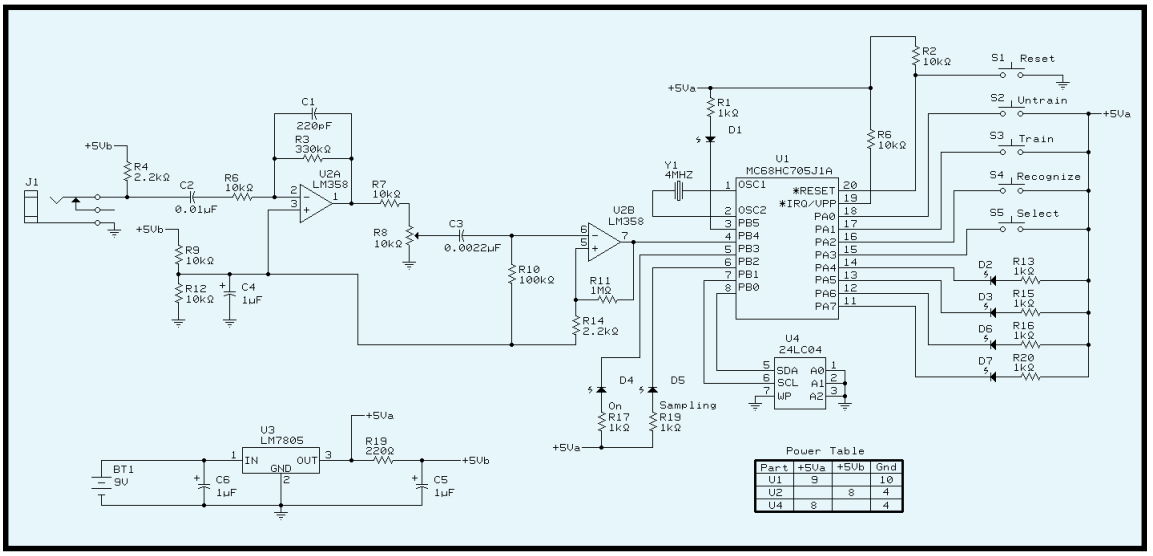

Figure 2—*An electret condenser microphone (not shown) is biased to 5 V via R4. The signal is then amplified by U2a. C2 and R6 (along with C3 and R10) form a high-pass filter. The output is fed to the second op-amp, which is configured as a comparator whose output is connected to PB4 of the 68HC705J1. The EEPROM has a two-wire I²C interface, which is connected to PB1 and PB0. The remaining pins of the processor are connected to LEDs and push buttons.*

uses ferroelectric cells.

It has several advantages over a more generic part. For one thing, the FRAM part can be written to over 10 billion times, compared to about 10k cycles with a generic EEPROM. This feature is important here because the first 128 bytes are used for scratch-pad memory and are constantly written to.

Also, it has a deep write buffer. So, once the starting address is specified, memory address is autoincremented and additional writes can be performed with no more intervention. As a result, writing to the device is very fast.

Generic parts, however, require you to set up the address every other byte before you write data. This task creates additional time overhead that may cause a bottleneck in the software flow—a major concern in a real-time system.

The FM24C04 has a low standby current of 25 µA as well as a low operation current of 100 µA. So, it's well suited for battery operation.

The EEPROM's first 128 bytes hold the transformed input utterance to be recognized or trained. Locations 128–512 store the feature vectors of a previously trained utterance. Each vector occupies 24 bytes, so the maximum number of templates that can be stored is 16.

The rest of the circuit comprises a 5-V regulator, switches, and LEDs.

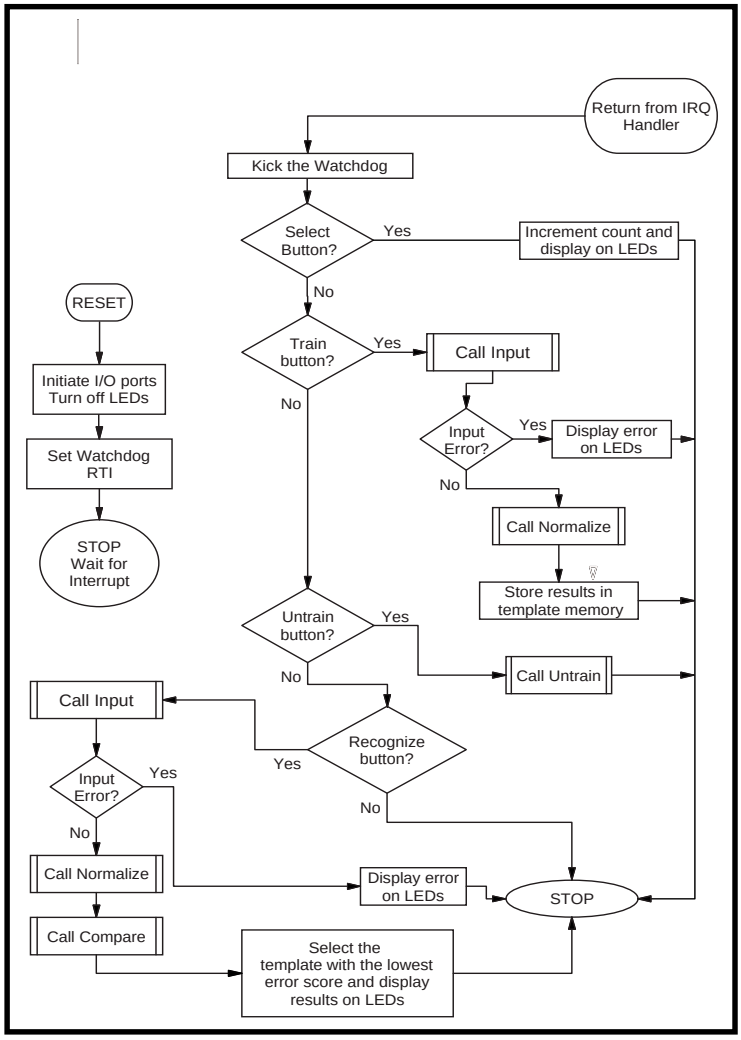

Figure 3—*The main routine performs the event handler. Events are generated by an interrupt caused by pressing a push button or by system reset. The events dispatched are Select, Train, Untrain, and Recognize.*

TINY USER INTERFACE

Before discussing the voice-recognition software, I want to describe the interface and how the system works from the user's point of view.

Seven LEDs and four switches compose the Tiny Voice user interface. LEDs D2, D3, D4, and D5 make up a four-bit binary number that gives Tiny Voice's status. It can either be the index of a voice command or an error message.

When power is connected or when the Reset switch is pressed, the Stop mode is entered. Pressing a push button activates the system and performs a certain function.

Pressing Select displays a binary number from 0 to 15 on four LEDs which selects the template number to be trained or untrained. Each time Select is pressed, the number increments to 15 and back to 0.

Pressing Train starts the Training mode. The On LED is activated, and the user is prompted to say the command to be trained.

While the user is speaking, the Sampling LED is lit during periods of speech and off during periods of silence. If the training is successful, the template is stored in EEPROM at the selected template location and the system enters the Stop mode.

Untrain modifies the data in the stored template so the pattern-matching algorithm skips over this template and does not consider it as a possible candidate.

This is useful for context switching of vocabularies. For example, out of the 16 templates, you may only need to scan for two words (e.g., "yes" or "no"), while ignoring the remaining 14.

To enable a template that was previously untrained, press the Train button and then press another button (e.g., Select) before speaking.

In Recognition mode, the speech is sampled and analyzed. The On LED is activated, and the user is prompted to say a previously trained command. As before, the Sampling LED is lit during speech and off during periods of silence.

The input is compared to the templates in memory and a decision made. If recognition is successful, the result is displayed on the four LEDs in binary.

When Reset is pressed, Stop mode is entered and the system is ready to accept a push-button command. Previously trained commands are not erased.

When an error occurs, the Error LED (D1) is lit and the error code is displayed in binary using the same four LEDs that display the template index number. After ~2 s, the LEDs go off and the system enters Stop mode.

The error codes—Time Out, Buffer Full, and Not Recognized—are defined in the header file.

After Train or Recognize is pressed, the system waits for valid speech input. If no input occurs after ~6.5 s, the system enters the Stop condition and the Time Out error code is displayed.

On the other hand, if the length of the utterance is longer than 1.6 s, the

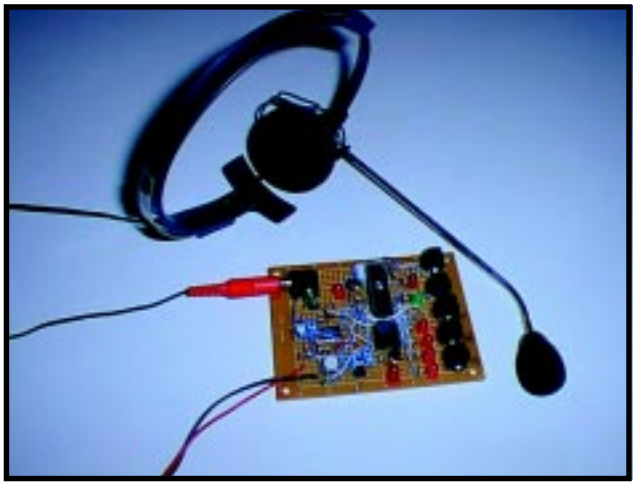

Photo 1—*My prototype was built on a 3″×3″ breadboard and is powered off a 9-V battery. The only ICs are the 68HC705J1 processor, LM358 dual-operational amplifier, the 4096-bit FM24C04 FRAM serial memory, and a 78L05 5-V regulator.*

system enters the Stop mode and the Buffer Full error is displayed.

The Not Recognized error code is displayed if the input utterance doesn't match a stored template. The system then enters Stop mode and waits for new input.

TINY ALGORITHMS

The software for Tiny Voice was written entirely in assembly. There is a total of eight routines.

The main program, MAIN.ASM, responds to events and schedules the remaining subroutines.

COMPARE.SUB handles the pattern matching. It compares the input template to each active template in memory and calculates the best match.

EEPROM.SUB handles the reading and writing of data to the EEPROM. It bit-bangs two I/O pins to simulate an I^2C protocol used by the EEPROM.

IRQ.SUB is the interrupt handler. Interrupts are caused by a button press.

The most complicated routine is INPUT.SUB. It samples the input, determines where the word starts and ends, and builds up the voice template.

TIME_NOR.SUB normalizes the length of the speech input to a fixed length of twelve two-element data values.

DIV16_8.SUB is an integer divide routine that divides a 16-bit number by an 8-bit number. This routine is called repeatedly by the time-normalization routine.

And finally, DELAYMS.SUB is a simple program where a delay is set by the value passed in the accumulator.

Tiny Voice is entirely event-driven and spends most of its time in the Stop mode. Events are caused by the interrupt of pressing push buttons. The event handler is shown in Figure 3.

INPUT ROUTINE

When a Recognize or Train event occurs, the input routine is invoked (see Figure 4). A timer is set up and polled until 110 µs has elapsed.

An interrupt routine could have been used to time the samples every 110 µs, but I was concerned that the overhead to service the interrupt might make it difficult to complete all the paths in the input routine within 110 µs.

Once the time elapses, the input square wave is sampled. If the sign changes from the previous measurement, one of the two frequency bytes is updated.

The threshold limit is set to six. In other words, if the pulse (positive or negative) is greater than six samples (roughly corresponding to 1.5 kHz), the "high" frequency byte is incremented by one. If it's less than six, the "low" frequency byte is incremented.

The rest of the routine is basically a state machine that uses speech activity as an input to determine a utterance bounded by silence. At each rising or falling edge, another byte counts the zero crossings.

After 256 samples, a frame counter advances and several tests are made. If the frame counter is greater than 64, the input buffer is filled (i.e., you spoke too long) or there is too much background noise, and an error is generated.

Otherwise, a timeout value is decremented and tested. This setup enables the routine to exit if too much time elapses before any sound is input.

If the buffer isn't full or a timeout has not occurred, then it tests the zero-crossing counter. Too low a value signifies silence, and a silence counter is incremented.

Otherwise, a sound-activity counter is incremented. If the sound-activity value is above a certain threshold and the silence value is high enough, the routine exits with a valid data sample.

TIME NORMALIZATION

Words vary in length. But for this algorithm to work, the lengths must be normalized to a fixed value.

Each sample consists of two bytes sampled over one frame of 256 samples. The unnormalized data in the first 128 bytes of the EEPROM is normalized to a set of 12 vectors in main RAM.

The vector in RAM is built up, element by element, by down- or up-sampling the raw data in EEPROM. Since there are two elements per feature, a template has a fixed memory length of 24 bytes.

THE MAIN ROUTINE

If the event is for training, the normalized vector in RAM is stored in memory according to the template number selected. Templates are stored in memory locations 128–512, which allows for sixteen 24-byte templates. No comparisons are performed.

If the system is recognizing, the normalized input utterance, which is stored in RAM, is compared element by element to each previously trained template stored in EEPROM.

The comparison is a simple Euclidean distance measure, and an error value accumulates. The minimum error value is selected and compared to a threshold.

If the result is above the threshold, the system rejects the recognition. If the value is low enough, the word is recognized.

Well, almost. Two more criteria must also be met: the score must be low enough, and the two smallest scores must differ by a large enough value.

TINY APPLICATIONS

For testing purposes, the system was trained with eight words: "VCR", "television", "telephone", "stereo",

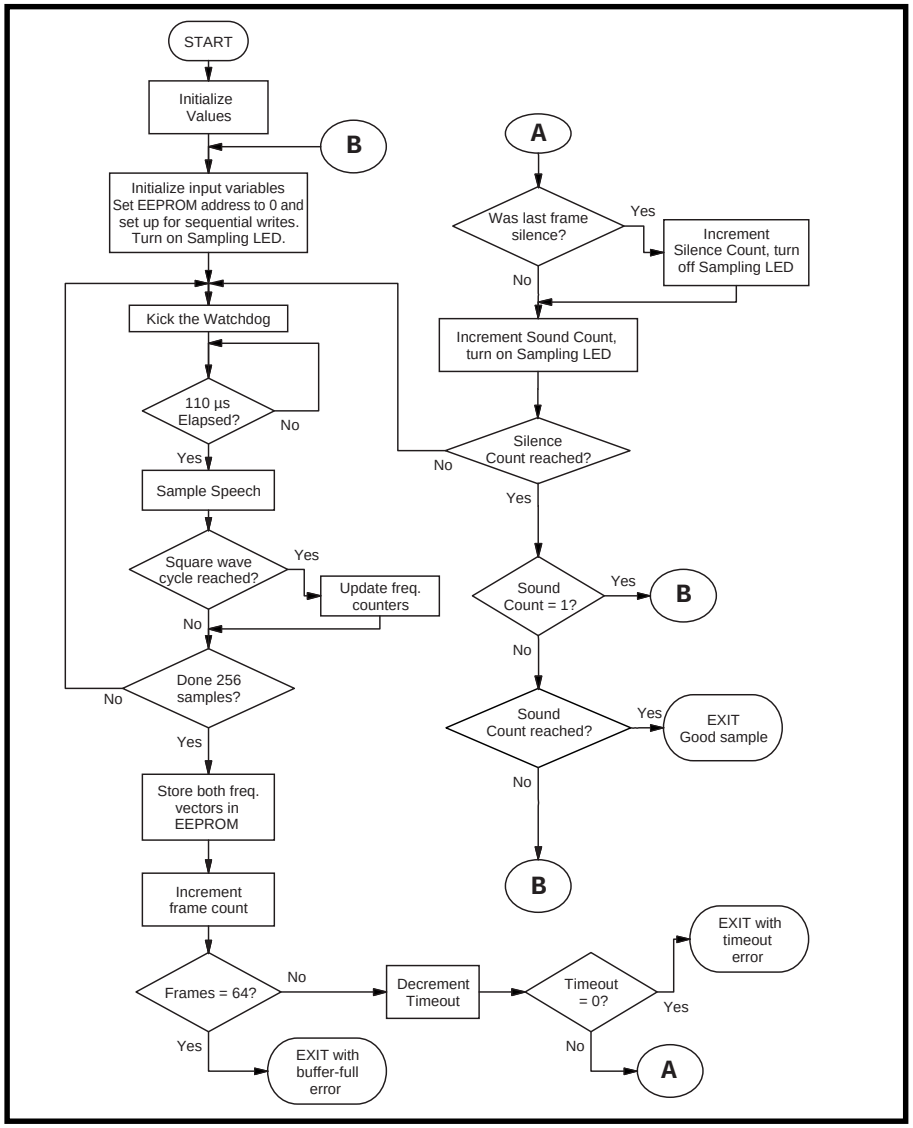

Figure 4—*Every 110 µs, the square-wave input is sampled and several options are considered, depending on the state of the frame, zero-crossing, silence, and sound counts. The state machine effectively captures the input utterance, while rejecting short bursts and input errors due to excessive background noise.*

"CD", "PC", "yes", and "no". Each word was trained twice, thereby occupying 16 templates.

Recognition accuracy approaches 100% when background noise isn't too severe. It also works with ~90% accuracy using speakers who didn't train the system.

A speaker-independent vocabulary can be constructed by having multiple trainings of a few words. For example, training "yes" and "no" eight times over a set of different speakers yields excellent results.

A note of caution: when using Tiny Voice, don't use a lot of short words (e.g., the numbers "one", "two", etc.). They're a bit beyond its capabilities.

And watch for commands that sound alike. For example, "on" and "off" will get you in trouble. Instead, try "turn on" and "off please".

A fun application might be a voice-activated padlock. Change the code so you have to enter one, two, or three voice commands in sequence. Then, multiply the scores. If the result is small enough, then "open sez me."

FUTURE TINY ENHANCEMENTS

Naturally, there are ways to improve the system. I was surprised by the HC05's speed. I also wound up with at least 200 bytes of leftover ROM for more code. Tiny Voice's code is modular, and updates can be easily added.

I can increase the EEPROM capacity to 1 or even 2 KB. This size would provide more template storage or allow for more frame features to better resolve differences in speech patterns.

I'd also like to add some fuzzy logic to the pattern-matching algorithm to improve recognition accuracy and the rejection criteria.

Adding a serial port instead of push buttons and LEDs could reduce cost and add more functionality. Threshold values could be changed, templates uploaded and downloaded, and so on.

I want an MCU-controlled gain adjustment on the input for different microphone levels and background noise.

Another improvement would be to add a dynamic time warp (DTW) algorithm to the pattern-matching routine. The DTW takes into account slight variations on how each word is pronounced—in particular, variations in lengths of phonemes.

But with only 200 bytes of code space left over, adding a DTW would be challenging. A first-order approximation may be achievable, however.

I'd rather use C than assembly language. When I started this project, I knew squeezing this functionality into 1200 bytes would be tough. So, a high-level language was out of the question.

Since then, I've had the opportunity to try out a C compiler from Byte Craft. The good news is, it generates small enough code. The bad news: I wish I'd used it earlier.

And as a final wish, I would like to use a different processor. Of all these improvements, this one is probably the best. You can now get equivalent MCUs with built-in ADCs, which would provide more elaborate signal processing and better noise rejection.

One of the best candidates for a low-cost system is the Sharp SM8500 8-bit MCU. It has almost everything you need for an embedded voice-command system, including a 10-bit ADC (8 channels) and an 8-bit DAC, which is useful for voice feedback and verification.

The SM8500 features SIO and UART ports to communicate with other system devices, 2 KB of internal RAM, as well as internal ROM and the ability to access external ROM or RAM. It also offers 80+ I/O pins for keypad and display interfacing, hardware multiply and divide, and a 250-ns instruction

cycle time. And, it costs under $3.

If you're willing to spend a bit more, then a new level of performance may be realized. New 32-bit RISC MCUs are becoming available in the sub $15 or even sub $10 range.

For example, the Sharp ARM710M RISC processor, running at a conservative 16 MHz, performs a complete FFT-Mel-Cepstrum analysis using only 50% of the processor's resources.

With the ability of RISC processors to address large amounts of memory, you have the ingredients to put together a dictation system like the one I'm using now. And, it can run off a couple pen-light batteries!

Brad Stewart is currently the product technical manager for RISC processors at Sharp Electronics. He also served as technical director for IPI, which specialized in voice-recognition and speech-compression software, and vice president of Covox, which specialized in multimedia products. You may reach Brad at bstewart@e-z.net or bstewart@sharpsec.com.

SOFTWARE

Source code (tinyvoice.zip) for this article may be downloaded from the Circuit Cellar Web site.

REFERENCES

B. Georgiou, "Give an Ear to Your Computer," *BYTE*, 56–91, June, 1978.

Motorola, *MC68HC05J1A Technical Data Manual*, 1997.

Sharp Electronics, *SM8500 User's Guide*, 1997.

B.C. Stewart and S. Sidman, "Design and Use of Voice Recognition in Embedded Applications," Paper presented at ESC East, Boston, MA, 1997.

SOURCES

68HC705J1A
Motorola
MCU Information Line
P.O. Box 13026
Austin, TX 78711-3026
(512) 328-2268
Fax: (512) 891-4465

FM24C02
Ramtron Intl. Corp.
1850 Ramtron Dr.
Colorado Springs, CO 80921
(719) 481-7000
Fax: (719) 481-9294
www.ramtron.com

C Compiler
Byte Craft Limited
421 King St. N.
Waterloo, ON
Canada N21 4E4
(519) 888-6911
Fax: (519) 746-6751
www.bytecraft.com

ARM710M, SM8500
Sharp Electronics Corp.
Microelectronics Gr.
5700 NW Pacific Rim Blvd., Ste. 20
Camas, WA 98607
(206) 834-2500
Fax: (206) 834-8903
www.sharpmeg.com