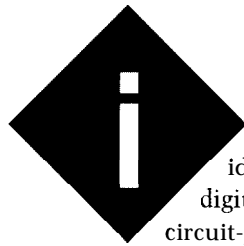


FEATURE ARTICLE

Sébastien Roy

Speech Compression Techniques and the CDV-1 Digital Voice Box

None of the methods for storing digitally recorded speech met project needs. After summarizing various methods, Sébastien refines the companded differential PCM method, a solution which fit the bill.



I started with the idea of designing a digital-voice-box-circuit-you know, one capable of storing and playing half a minute of digital sound. The list of applications (e.g., talking posters, door bells, etc.) is endless for a device that stores sounds in a small physical space. For instance, imagine greeting a trespasser on your property with a loud, "No trespassing. Be gone Intruder! "

I wanted the device to be commercially viable. The features and cost needed to be competitive with similar products. Most importantly, the device had to be as small as possible.

With these considerations in mind, I came up with a set of specifications. It needed:

- a sound coding method to achieve 2: 1 compression over 8-bit PCM (with this, a 64-KB EPROM to hold 16 s of sound sampled at 8 kHz),
- to generate only frequencies below 1.7 MHz, eliminating the need for FCC or DOC approval (DOC is the Canadian equivalent to the FCC),
- to be compact for use in space-conscious applications, and
- sound creation for the voice box using a personal computer without additional, expensive hardware.

Digital sound data contains a lot of redundancy and is compressed in a number of ways. My challenge was to find a compression method powerful enough to produce a 2:1 gain and simple enough for a microcontroller to run on 1.7 MHz.

I decided to let the microcontroller handle the signal processing tasks without specialized hardware. I wasn't sure this was possible, but dedicated speech processing chips significantly add to project cost and impose a specific coding scheme such as ADPCM or CVSD. Also, I wanted to find out how much a low-end microcontroller could accomplish.

At that point, I was pretty sure I had to sacrifice sound quality to meet these constraints. Nonetheless, I set out to design the perfect voice box: compact, cheap, elegant, simple, and "FCC proof." I was in for surprises on the long road to a prototype.

THEORY OF SPEECH COMPRESSION

First, I reviewed the theory of speech compression methods and algorithms to find something that would fit the bill. My research and reflection process produced interesting insights. I eventually created a new speech-compression algorithm. I'll start by outlining the essential concepts of speech-coding theory.

The advantages of representing sound or speech digitally are well known. Digital sound representation makes possible encryption for privacy. Digital data is more resistant to degradation than analog recordings and is easier to manipulate. Finally, error-correction systems work only with digital sounds for transmission over a noisy channel.

Conventional analog-to-digital conversion techniques create large digital files. For example, in digital telephony the bandwidth ranges from 0.3 to 3.3 kHz. The sampling rate is usually 8 kHz with a resolution of 12 bits. Accordingly, a single second of telephone-quality speech can occupy as much as 96,000 bits or 11.7 KB.

However, speech signal redundancy makes it possible to encode speech more efficiently. The compres-

sion method must be carefully selected to provide an adequate balance between sound quality, compression ratio, and computational complexity. If we choose the signal-to-noise (S/N) ratio as a measure of sound quality, this equation is handy:

$$S_{dB} = 6B + 4.8 - 20\log_{10}h \quad (1)$$

where S_{dB} is the S/N ratio in decibels, B is the number of bits per sample, and h is the headroom factor (i.e., a safety margin to prevent saturation or clipping, usually set to 4). This formula confirms that 12 bits are required for a S/N ratio of 60 dB with a headroom factor of 4.

Speech signals present a coherent structure, which lends to compression efforts. Two characteristics are useful for simple compression algorithms.

First, speech is concentrated in the low amplitudes, which means that statistically the smaller sample values occur more often than the larger ones. Compression methods like companding and Huffman encoding take advantage of this characteristic.

Second, there is a high correlation between neighboring samples because speech parameters (pitch, amplitude, etc.) vary slowly in time. ADPCM and CVSD take advantage of this.

It is important to distinguish between speech encoders and speech transcoders. Speech encoders convert analog speech into a given digital representation. Coding schemes like ADPCM and CVSD work directly from the analog signal.

Speech transcoders convert digitally encoded speech to another form of digital speech. For example, Huffman encoding requires a PCM digital sound file as input.

Now, let's move on to look at various types of compression.

COMPRESSION BASED ON SPEECH DENSITY

Speech signals are heavily concentrated in the low amplitudes, which means the probability density of speech is not uniform. In fact, speech approximates a modified gamma density function. Hence, a uniform A/D quantizer does not yield the optimal S/N ratio. It is much better to

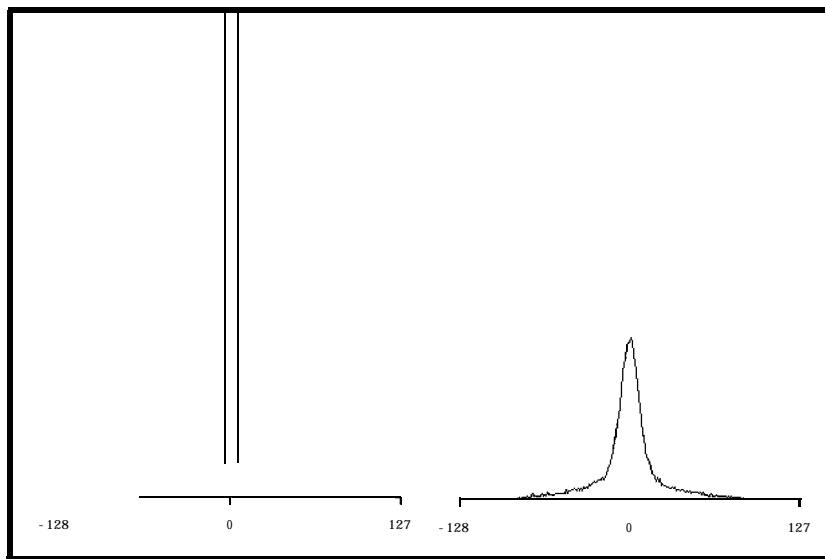


Figure 1—The probability density functions of DPCM (left) and PCM (right) are derived from a small sound data set. Observe that the concentration in the low amplitudes is much more pronounced in DPCM than in PCM.

use a nonuniform quantizer in which the steps are smallest at lower levels. The step sizes in the optimal quantizer are adjusted to make each sample value equally likely.

A nonuniform quantizer usually consists of a nonlinear distortion filter followed by a uniform quantizer. The filter compresses the dynamic range of the signal to make its probability density as close to uniform as possible. p-law and A-law are the two distortion curves often used for these filters.

The process is called *companding* (for compressing/expanding) and is heavily used in digital telephony. Standard 8-bit u-law compander chips produce roughly the same sound quality as a 12-bit uniform quantizer.

Companding gives a compression ratio of roughly 3:2 and its principles are fairly straightforward. But, how can it be applied to our voice box? The obvious way is to incorporate a dedicated 8-bit u-law codec chip to act as a nonlinear D/A converter.

This solution, however, has two drawbacks. Codec chips tend to be expensive. In addition, they are designed to give 12-bit quality in 8 bits. What we really want is 8-bit quality in 4 bits! (Some codecs compress 8 bits into 6 bits, but they're still expensive.)

Of course, it is possible to implement companding in software with little overhead. However, while

companding holds promise, it does not reach the target 2:1 compression ratio by itself.

Huffman encoding uses variable-size words to represent samples. Short 2-, 3-, or 4-bit words represent the most frequent values while longer (up to 12-bit) words represent the rarest values. Unfortunately, this method is too complex for our purposes because the words are of unequal size.

COMPRESSION BASED ON AUTOCORRELATION

Most popular voice-compression schemes take advantage of the high correlation between successive samples in speech signals. For example, differential PCM (DPCM) encodes the difference between samples instead of the samples themselves.

While this method does not offer significant compression, its data has a much lower average power than standard PCM. In other words, a DPCM data set should be easier to compress than the equivalent PCM data set due to its reduced variance.

Incorporating a prediction filter in the encoding process improves the basic DPCM concept. Such a filter embodies knowledge of the statistical properties of speech signals in its coefficients and predicts the value of an unknown sample based on previous samples.

The difference between a sample and its predicted estimate is encoded.

Using a second-order predictor, the variance of the system is reduced by 6 dB. According to Equation 1, a reduction of 6 dB implies that we can use one less bit than PCM in quantizing the residual and still maintain the same S/N ratio.

We can improve the performance of the prediction filter by making it adaptive. The resulting speech compression algorithm, adaptive DPCM (ADPCM), is widely used in telephony and computer-based speech.

If a little sound degradation is acceptable, ADPCM provides a compression ratio of 2:1, but it is complex. ADPCM is typically implemented using dedicated ICs or DSPs.

As far as our voice box is concerned, any type of prediction scheme is difficult to implement in real time because of the numerous multiplication operations involved.

Simpler differential modulation schemes include delta modulation (DM), which is really DPCM with a 1-bit quantizer. While this is an easy modulation scheme to implement, it is

Interval	CDPCM Symbol	Decoded DPCM Value
-128 to -19	0	-40
-18 to -10	1	-13
-9 to -6	2	-7
-5, -4	3	-4
-3	4	-3
-2	5	-2
-1	6	-1
0	7	0
1	8	1
2	9	2
3	10	3
4, 5	11	4
6-8	12	7
9-14	13	13
15-24	14	19
25-127	15	40

Table 1-A standard CDPCM look-up table is used for decoding.

difficult to obtain acceptable speech quality with DM. Even at a sampling rate of eight times the Nyquist frequency, the S/N ratio is only 20 dB. In other words, even at bit rates comparable to PCM, the sound quality leaves much to be desired.

Adaptive delta modulation schemes fare much better by adapting

the quantizer's step size as a function of recent sample values. Continuously Variable Slope Delta (CVSD) modulation is an adaptive system which uses a 1-bit quantizer.

A sample value of 1 means the output should be increased by the current step size while a 0 means the output should be decreased accordingly. CVSD attempts to vary the step size to minimize the occurrence of slope overload and granular noise.

Slope overload occurs when the slope of the analog signal is so steep that the encoder can't keep up. This effect can be minimized or prevented by enlarging the step size sufficiently.

Granular noise occurs when the analog signal is constant. The CVSD system has no symbols to represent steady state, so a constant input is represented by alternating ones and zeros. Accordingly, the effect of granular noise is minimized when the step size is sufficiently small.

The adaptation strategy used by CVSD is fairly simple. The previous three samples are examined. If they are

RTC-HC11

PROCESSOR BOARD

Offering an exceptional value in a single-board embedded controller, Micromint's RTC-HC11 combines all of the most-asked-for features into a compact 3.5" x 4.5" package at a reasonable price. Featuring the popular Motorola MC68HC118-bit microcontroller, the RTC-HC11 gives you up to 21 lines of TTL-compatible I/O; an 8-bit, B-channel analog-to-digital converter; two serial ports; a real-time clock/calendar with battery backup; 512 bytes of nonvolatile EEPROM; and up to 64K of on-board RAM or EPROM, 32K of which can be battery backed.

Software development can be done directly on the RTC-HC11 target system using BASIC-11, an extremely efficient integer BASIC interpreter with dedicated keywords for I/O port, A/D converter, timer, interrupts, and EEPROM support. In addition, a flexible configuration system allows a BASIC program to be saved in the on-board, battery-backed static RAM, and then automatically executed on power-up. Micromint also offers several hardware and software options for the RTC-HC11 including the full line of RTC-series expansion boards as well as an assembler, ROM monitor, and a C language cross-compiler.

Additional features include:

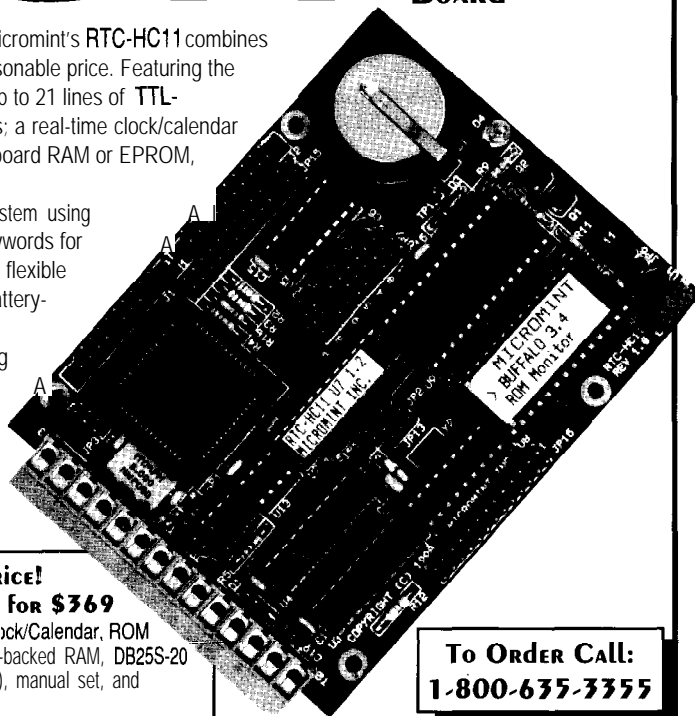
- Asynchronous serial port with full-duplex RS-232 and half-duplex RS-485 drivers
- 1-MHz synchronous serial port
- CPU watchdog security
- Low-power "sleep" mode
- 5-volt-only operation
- RTC stacking expansion bus

Special DEVELOPMENT SYSTEM Price!

RTCHC11-DEV A \$477 value for **\$369**

Board w/ 8-bit ADC, EEPROM, 8K RAM, Clock/Calendar, ROM monitor, BASIC-11 in EPROM, 32K battery-backed RAM, DB25S-20 serial cable, utilities diskette (PC compatible), manual set, and HCTerm software.

Other configurations starting at \$239



**To Order Call:
1-800-635-3355**



MICROMINT, INC. 4 Park Street • Vernon, CT 06066 • (203) 871-6170 • Fax (203) 872-2204
in Europe: 0285-658122 • in Canada: (514) 336-9426 • in Australia: (3) 467-7194 • Distributor Inquiries Invited!

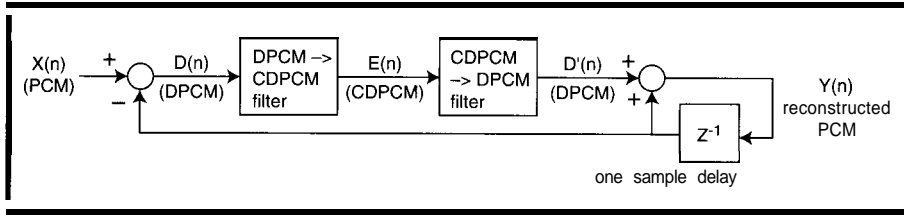


Figure 2—The CDPCM encoder uses the reconstructed PCM signal $Y(n)$ to avoid cumulative errors in the CDPCM datastream $E(n)$.

identical, it indicates slope overload. The step size is therefore increased by a multiplicative constant and an additive constant.

If the three samples are not identical, the step size is progressively reduced. It is important to carefully choose the values of the constants used to adjust the step size so the effects of both slope overload and granular noise are minimized.

To achieve adequate speech quality with CVSD, the sampling frequency should be four times higher than with PCM. Consequently, CVSD systems often work at 32 kbps, which is half the bit rate required by an 8-bit, 8-kHz PCM system.

CVSD proves that a relatively simple algorithm can achieve significant compression gains over standard

PCM. Nonetheless, there are a number of reasons why CVSD does not fit the design constraints outlined earlier.

CVSD requires a lot of bit-level manipulation. It also involves some multiplication in the step-size adjustment procedure. Despite its conceptual simplicity, CVSD might still be too complex to be implemented on a simple controller running at 1.7 MHz.

Furthermore, the sound quality is better if the analog signal is directly coded into CVSD. The constraints outlined in the beginning specify the use of a PC to create the sounds, so the analog signal is not available for direct coding.

Of course, a simple software transcoder can be

written to convert between the PCM data typically produced by computer sound hardware and CVSD. However, the transcoding process results in a loss of quality because the quantization noise introduced by PCM is compounded by the noise inherent to CVSD.

Indeed, PCM introduces quantization noise because it is limited to a discrete number levels—256 in the case of an 8-bit system. CVSD has no such limitation. In fact, dedicated CVSD chips typically use 10 bits internally to represent voltage levels.

However, CVSD introduces slope overload and granular noise because of its adaptive nature. When using a transcoder, both PCM and CVSD noise are present and their combined effects further deteriorate sound quality.

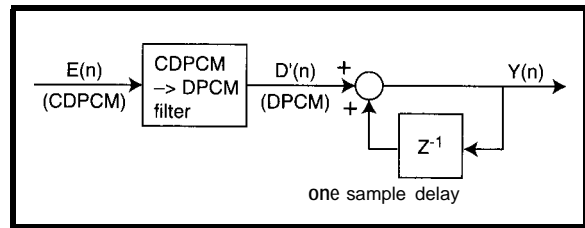


Figure 3—The CDPCM decoder is simple—only a look-up table and an addition are necessary to decode a sample.

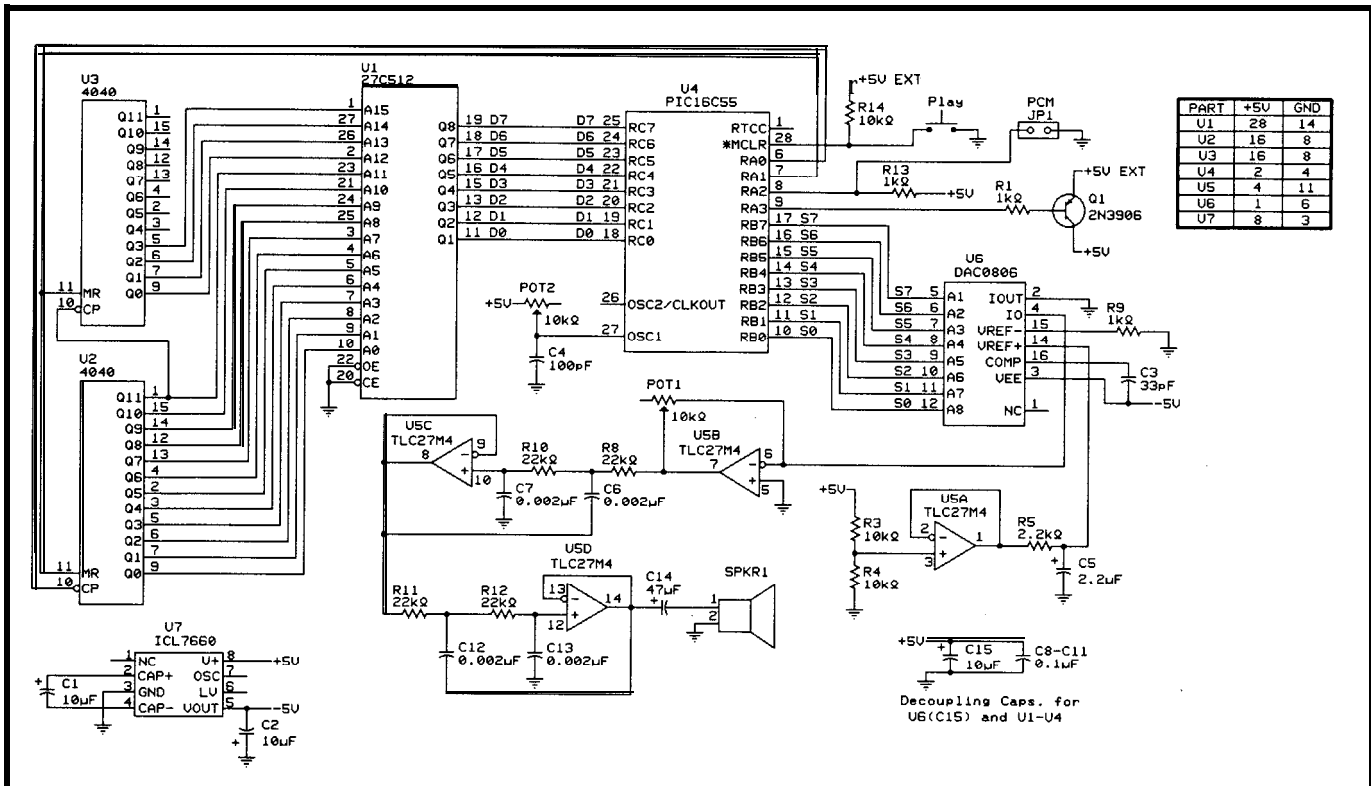


Figure 4—The CDV-1 is composed of a PIC16C55 microcontroller, an EPROM, counters, a D/A converter, and a simple analog section that provides amplification and filtering.

Having examined all these speech digitizing methods, it became painfully obvious to me that none of them, not even CVSD, met the constraints of the voice-box project.

However, I remained convinced that sound signals contain enough inherent redundancy to provide a solution. It seemed possible to borrow concepts from existing speech compression-methods to create a new method specifically for this project.

COMPANDED DIFFERENTIAL PCM

I needed an algorithm that exploited redundancy in sound signals and required no more than a couple of additions and/or table lookups to decode a sample. Only companding seemed simple enough to implement in software. Unfortunately, its compression ratio is less than satisfactory.

The probability density curves in Figure 1 reveal some statistical aspects of a differential PCM data set compared to ordinary PCM. DPCM exhibits more redundancy. Of the 256 possible values, 16 account for over 65% of the samples. It should therefore be possible to compress a DPCM data set more efficiently than PCM using Huffman encoding or companding.

But how do we apply a concept like companding to differential data? As you may recall, p-law and A-law companding distort the sound signal giving it a uniform probability density. The goal is to make each sample equally likely.

I propose a new speech compression method called **CDPCM** (Companded Differential PCM) which maps the 256 possible DPCM sample values to 16 equally likely symbols. The computer generates a DPCM probability curve based on a large, representative speech data set to construct a standard look-up table.

The curve is divided into 16 intervals of equal likelihood. Each interval is mapped to one of 16 symbols, which means only 4 bits are required to encode a sample. We thus achieve 2: 1 compression.

A symbol is decoded by replacing it with the most likely value in the interval it represents (see Table 1). This introduces error in the recon-

Listing 1—This pseudocode works through the algorithm for the playback of PCM with oversampling.

```

1  read a sample from the EPROM
2  is it equal to 255? If so, go to sleep
3  copy sample to 255? If so, go to sleep
4  increment counters to update EPROM address
5  wait 14 cycles
6  take current value on D/A converter port and add it to
   next sample value from EPROM
7  divide by two using the "rotate right" instruction
8  copy this estimated sample to D/A converter
9  wait 14 cycles
10 go back to step 1

```

structed speech signal, but the overall impact on speech quality is minimal.

While there are only 16 symbols, the reconstructed PCM samples from a CDPCM decoder can take any value from 0 to 255. The difference between adjacent samples is limited to 16 values.

A detailed statistical analysis of the noise introduced by this algorithm is beyond the scope of this article. Suffice it to say that CDPCM-induced noise concentration is in the high frequencies and can be virtually eliminated by playing it through an appropriate filter.

When a decoded CDPCM signal plays back without a filter, noise appears only for plosive sounds (e.g., if you say "p," the burst of air on your hand marks it as a plosive sound). When it plays back through an appropriate low-pass filter, the difference in quality with PCM is nearly inaudible.

The process of reconverting data to PCM at the decoding end is like an integrator. Accordingly, any error we introduce in the DPCM data is integrated to infinity. The encoder tracks the error introduced and compensates, avoiding this problem. This is accomplished by using the reconstructed PCM signal $Y(n)$ (the signal obtained by decoding the CDPCM data set) in the encoding process (see Figure 2).

Ordinarily, the DPCM signal $D(n)$ is derived by using:

$$d[n] = x[n] - x[n-1] \quad (2)$$

The CDPCM signal $E(n)$ is obtained simply by passing $D(n)$ through the look-up table. At the decoding end, the

reconstructed DPCM signal $D'(n)$ is likewise obtained by passing $E(n)$ through the opposite look-up table. This equation produces the decoded PCM output:

$$y[n] = y[n-1] + d'[n]$$

To compensate for error, $Y(n)$ is used as a reference for deriving $D(n)$. Equation 2 must then be replaced by:

$$d[n] = x[n] - y[n-1]$$

This equation implies that the encoder includes a simulated decoder. In fact, as Figure 3 indicates, the encoder is more complex than the decoder. Only the decoder needs to be embedded in the voice box.

The encoder must still address the possibility of overflow. The error introduced by the encoder may bring $Y(n)$ above 256 or below 0. The possibility of overflow must be verified by the encoder and dealt with by replacing the offending symbol by a smaller one.

DIGITAL VOICE BOX

The CDV-1 circuit is made up of a microcontroller with the CDPCM decoding algorithm embedded in its firmware, an EPROM containing the actual sound data, a standard 8-bit D/A converter, and a simple analog filter and amplifier circuit.

You only need an external S-V power source, a push-button switch or TTL activation signal; and a speaker. Activating the switch triggers the playback of the sound data. At the end, the circuit goes into low-power sleep mode until the next request.

Listing 2—A simple C program generates a raw CDPCM file from PCM data contained in an AIFF file.

```
#include <stdio.h>

unsigned char encode(int *d);

main()
{
    FILE *old, *new;
    int c, i, d, mo;
    unsigned char nibble[2];
    char table[16] = {-40,-13,-7,-4,-3,-2,-1,0,1,2,3,4,7,11,19,40};
    old = fopen("filename of source (PCM) sound file", "rb");
    new = fopen("CDPCM Output filename", "wb");

    /* The following section should be used only with AIFF files */
    i = 0;
    while (i == 0) {                /* finding the 'SSND' chunk */
        c = fgetc(old);
        if (c == 'S') {
            c = fgetc(old);
            if (c == 'S') {
                c = fgetc(old);
                if (c == 'N') {
                    c = fgetc(old);
                    if (c == 'D') {
                        c = fgetc(old);
                        i = 1;
                    }
                }
            }
        }
    }
    for (i = 0; i < 12; i++) { /* skip the header */
        c = fgetc(old);
    }
    /* End of AIFF section */
    mo = 128;                /* initial value of the running sum is 0=20
                             in offset binary, that becomes 128 */
    i = cnt = 0;
    while ((c = fgetc(old)) != EOF) {
        if (c < 128)                /* Convert 2's complement to */
            c += 128;                /* binary offset */
        else
            c -= 128;

        d = c - mo;                /* Generate DPCM sample */
        nibble[i] = encode(&d);
        while (mo + d > 255)        /* correct overload conditions */
            d = table[--nibble[i]];
        while (mo + d < 0)
            d = table[++nibble[i]];
        if (i == 0)                /* Combine two nybbles, output a byte */
            i = 1;
        else {
            i = 0;
            fputc((unsigned char)((nibble[1] << 4) | nibble[0]), new);
        }
        mo += d;                /* update running sum */
    }
    fputc(EOF, new);                /* close files */
    fclose(old);
    fclose(new);
}

/* Convert from DPCM to CDPCM by converting values within specific
intervals to one of sixteen equally likely symbols. The value of d
is modified to serve as reference in the main loop and thus compensate
the cumulative effect of errors */
```

(continued)

\$149 4A C Compiler?

You heard right! A quality K&R C compiler designed for the 8051 microcontroller family, just \$149, **including** the Intel compatible assembler and linker. A source level simulator is also available for just \$169. A great companion to our fine Single Board Computers, like those below. **CALL NOW!**

552SBC

80C552 a '51 Compatible Micro
40 Bits of Digital I/O
8 Channels of **10** Bit A/D
3 Serial Ports (RS-232 or 422/485)
2 Pulse Width Modulation Outputs
6 Capture/Compare Inputs
1 Real Time Clock
64K bytes Static RAM
1+ UVPROM Socket
512 bytes of Serial EEPROM
1 Watchdog
1 Power Fail Interrupt
1 On-Board Power Regulation
1 Expansion Bus

Priced at just **\$229** in 100 piece quantities. Call about our 552SBC C Development Kit, just \$499 qty 1.

Hyperactive '51!

Our popular 8031SBC can now be shipped with Dallas Semi's hyperactive **DS80C320**, an 8051 on steroids. Averaging 3x faster than the standard 51, your project can really scream! **Call or ftp for pricing and brochures today!**

Other versions of the 8031SBC have processors with on-chip capture registers, EEPROM, IIC, A/D and more. Call or ftp for a list!

8031SBC as low as **\$49**

Call for your custom product needs. Quick Response.

HTE HiTech Equipment Corp.
 9400 Activity Road
 San Diego, CA 92126
 [Fax: (619) 530-1458]

Since 1983

(619) 566-1892



Internet e-mail: info@hte.com
 Internet ftp: ftp.hte.com

The microcontroller choice was not difficult. The 87C751 was a serious candidate, but the only low-cost micro capable of handling the required load with a clock speed of 1.7 MHz was the PIC16C55. This suitability is largely because of the RISC design of the PIC family.

Most instructions execute in only one clock cycle. Because the timing of the instructions is so uniform, it also facilitates development of time-critical firmware. Absolutely indispensable for this project is the PIC's unique SWAP instruction, which interchanges the high and low nybbles within a byte in one clock cycle.

CDV-1 FEATURES

If the microcontroller is running at 1.376 MHz, the CDV-1 is capable of playing back digitized speech at a sampling rate of 8 kHz and a resolution of 8 bits while oversampling. Indeed, the firmware performs interpolation to bring the effective oversampling rate to 16 kHz. Besides improving sound quality, this interpolation also eases filter requirements.

The circuit uses a 27C5 12 EPROM which holds either CDPCM data or standard PCM data. A jumper on the board provides the desired sound format. The total current consumption is less than 50 mA during playback and less than 4 μ A in sleep mode. The circuit works with any 4-6-V supply.

An RC oscillator clocks the PIC microcontroller to keep the cost down. With this type of oscillator, the operating frequency varies significantly from one PIC to another.

An adjustable RC oscillator with a 10-k Ω potentiometer is incorporated in the design because the sound's playback speed is a function of the oscillator frequency. The user can thus adjust the operating frequency from 0.7 to 2.5 MHz until the playback speed is adequate. This feature enables the playback of sounds sampled at more than 8 kHz (e.g., up to around 14 kHz).

Adjust the oscillator for adequate playback of 8-kHz sounds (around 1.4 MHz) and then fix it in place by taping the potentiometer or by gluing it with epoxy. The CDV-1 also includes a volume control potentiometer in the

Listing 2—continued

```

unsigned char encode(int *d)
{
    unsigned char res;

    switch (*d){
        case -5:
            case -4: *d = -4; res = 3;
                    break;
            case -3: res = 4;
                    break;
            case -2: res = 5;
                    break;
            case -1: res = 6;
                    break;
            case 0:  res = 7;
                    break;
            case 1:  res = 8;
                    break;
            case 2:  res = 9;
                    break;
            case 3:  res = 10;
                    break;
            case 4:
            case 5: *d = 4; res = 11
                    break;
        default:  if (*d <= -19){res = 0; *d = -40;}
                    else
                        if (*d <= -10 && *d > -19){res = 1; *d = -13;}
                            else
                                if (*d <= -6 && *d > -10) {res = 2; *d = -7;}
                                    else
                                        if (*d <= 8 && *d > 5){res = 12; *d = 7;}
                                            else
                                                if (*d <= 14 && *d > 8){res = 13; *d = 11;}
                                                    else
                                                        if (*d <= 24 && *d > 14){res = 14; *d = 19;}
                                                            else
                                                                if (*d >= 25){res = 15; *d = 40;}

                    break;
    }

    return(res);
}

```

analog section which should be adjusted carefully during playback to get maximum output power without saturation. You should be able to get up to 0.5 W with an 8- Ω speaker and up to 1 W with a 4- Ω speaker.

HARDWARE OVERVIEW

The CDV-1 (see Figure 4) is built on a 2 $\frac{5}{8}$ " x 2 $\frac{1}{16}$ " circuit board. (The board has recently been redone on a 2 x 3 $\frac{1}{2}$ " format.) It is as compact as possible for assembly. The ICs are in DIP packages and are relatively inexpensive and easy to find.

The microcontroller reads encoded sound data from the EPROM through port C and sends its output to the DAC0806 D/A converter through port

B. Two 4040 12-bit ripple counters are cascaded to generate the addresses for the EPROM. The EPROM scan rate is controlled by the microcontroller through line RAO which is tied to the counters' clocks.

RA3 controls transistor Q1, which turns power on and off to the rest of the circuit. When the CDV-1 is not active (e.g., it is not playing a sound), the microcontroller is in low-power sleep mode, and the other components of the circuit are completely powered off for minimum current consumption. A low level on the PIC's reset line wakes it up, turns on Q1, and triggers the sound-playback sequence.

The state of line RA2 tells the microcontroller whether to work in

PCM or CDPCM mode. R15 is tied to +5 V making CDPCM the default operating mode. To operate in PCM mode, a wire jumper must be added to the circuit board between RA2 and a nearby ground trace.

The DAC0806 is a garden-variety D/A converter, widely available and inexpensive. It requires a negative reference voltage provided by the inverter made up of U7, C1, and C2.

The D/A converter's analog output is piped to a three-stage low-pass filter/amplifier circuit built around a TLC27M4 quad op-amp. Its efficient, low-power, rail-to-rail operation and the fact that it operates from a 5-V supply make the TLC27M4 an ideal choice for this application.

The first stage consists of U5b, R6, and potentiometer R7. It serves as an adjustable amplifier controlling the sound output's volume. The first stage also converts the D/A converter's current output to voltage.

The second and third stages are two second-order low-pass filters with cut-off frequencies of 3.62 kHz. Together, they form an efficient fourth-order low-pass filter. Because the software performs oversampling, this is more than adequate to handle quantization noise. Finally, U5a provides a stable, filtered, 2.5-V reference to the D/A converter.

THE FIRMWARE

The firmware includes 132 carefully chosen instructions and handles the playback of PCM or CDPCM data in real time. The initialization code turns the circuit on, resets the counters, and verifies the state of line RA2 to determine whether it should jump to the PCM or the CDPCM code section.

The PCM section reads bytes from the EPROM and directly outputs them to the D/A converter. Between each sample pair, an estimated sample is computed. Listing 1 shows the broken-down algorithm in pseudocode.

The PCM data must be in "offset" format (not two's complement) be-

cause the software copies the samples directly from the EPROM to the D/A converter. Furthermore, the data must not contain the value 255 because it indicates the end of the sound data and shuts off the circuit.

If the sound has been properly recorded (i.e., the data has not been "clipped" because of excessive volume), it is highly unlikely that it contains a 255. Otherwise, a software filter can be written to replace all occurrences of 255 by 254.

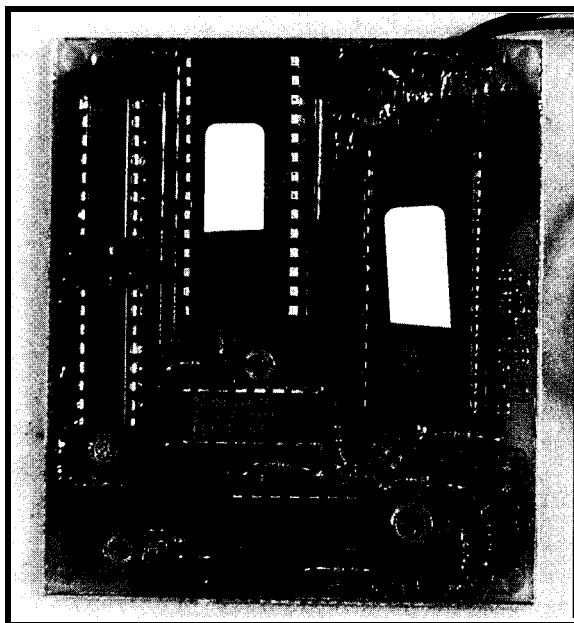


Photo 1—The CDV-1 board is about as small as it can get using DIP-style components.

The CDPCM code section is more involved. The code must be very tight to provide real-time playback of 8-kHz sounds at a clock speed of only 1.376 MHz. This is possible because CDPCM offloads the bulk of the work to the encoder.

If the micro detects an overflow condition in CDPCM mode, it assumes the sound data has ended and goes to sleep. Properly encoded CDPCM data never causes an overflow, but a string of 255s in the blank portion of the EPROM will.

PREPARING SOUND DATA

How do we create sounds and incorporate them in the CDV-1 voice box? The simplest way is through a computer capable of recording sounds (e.g., most Mac models, PCs with sound cards, etc.).

You can also get prerecorded sound files off a BBS or the Internet. It should then be easy to convert these samples to PCM format. If your software has an uncompressed sound format, it is probably PCM. Sun workstations are a special case because their sound hardware includes a u-law compander.

Sound software use WAV, AIFF, AU, SND and other file formats to store sounds. You need to find out enough about the sound format you are using to remove headers and other irrelevant information, leaving only the PCM data.

Then you need to know whether the PCM data is in two's complement format or offset binary. There are some interesting freeware and shareware offerings out there that enable you to modify, edit, and convert sounds between formats.

Listing 2 shows how a simple CDPCM encoder can be written in C. It is written on a Macintosh running Think C, but it should compile without modifications on most platforms.

The software is designed to work with AIFF files, a sound file format commonly used on Macs. The AIFF format is quite complex and involves a number of "chunks." The initialization portion scans the file from the beginning until it finds the header indicating the start of the sound data chunk. It then skips the header and encoding begins.

Each byte converts from two's complement to offset binary before being encoded as a 4-bit nybble. If a nybble causes an overflow, it is reduced until the situation is rectified. Nybbles are combined into bytes and written to the output file.

The file can then be programmed into an EPROM and incorporated in the CDV-1 circuit. Make sure the circuit is set to CDPCM mode (no jumper), plug in the batteries, 8- Ω speaker, and switch, and you're ready to hear the box talk. You also need to adjust both potentiometers for proper volume and pitch.

CONCLUSION

I hope seeing how some fancy theoretical concepts can be applied to a simple, practical project has been helpful. I also meant to illustrate that some real-time signal processing tasks can be handled by a lowly PIC micro running at 1.4 MHz.

When I started working on this project, I wanted to know if it was possible to decode CVSD in firmware with an inexpensive micro. I ended up doing something entirely different and much more rewarding. Sometimes, R&D is like gambling—you follow your hunch without knowing where it will lead you. This time I hit pay dirt. □

Sébastien Roy holds an M.Sc. in electrical engineering from l'Université Laval, Quebec City. He specializes in embedded systems, signal processing, and practical applications of the Internet. He offers consulting and design services under the umbrella of Tertius Technologie Inc. He may be reached at roys@nbnet.nb.ca.

REFERENCES

Speech encoding:

Jayant, N.S. "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," *Proc. IEEE*, **62**, No. 5, 61 1-632, 1974.

Jayant, N.S. and P. Noll. **Digital Coding of Waveforms**, Prentice-Hall, Englewood Cliffs, 1984.

Oliver, B.M., et al. "The Philosophy of PCM," *Proc. IEEE*, **36**, No. 11, 1324-1331, 1948.

CVSD and Predictive Coding:

Greefkes, J.A. "A Digitally Companded Delta Modulation Modem for Speech Compression," *Proc. IEEE Int. Conf. on Communications*, **7.33-7.48**, June 1970.

Markel, J.D., and A.H. Gray, Jr. **Linear Prediction of Speech**, Springer-Verlag, New York, 1976.

Makhoul, John. "Linear Prediction: A Tutorial Review," *Proc. IEEE*, **63**, No. 4, 1975.

SOURCE

Tertius Technologie, Inc.
RR #2, Site 17, Box 9
Beresford, NB
Canada EOB 1H0
(506) 542-1618

CDV- 1 kit includes documentation, software, sample sound files, PCB, programmed PIC, other ICs, and discretes.

Speaker, power supply, reset switch and EPROM not included.....\$29.95

CDV-1 A&T unit . \$32.95

Programmed PIC.....\$10

Programmed EPROM.....\$10

PC board, no silkscreen . . . \$12

All prices in U.S. dollars. Shipping extra. Call for OEM pricing.

IRS

404 Very Useful

405 Moderately Useful

406 Not Useful

don't like windows?
don't care for r&d?
sorry.

Then we just can't help you. But if you're looking for a high-capacity, user-friendly EDA system, we've got just what you need. Say "hi" to EDWin, your new companion in Electronics Design. EDWin features seamless integration between modules, so you can finally kiss the tedious concept of front- and back annotation goodbye. EDWin gives you all the tools you'll need, and is so user-friendly you can even compile your own custom toolboxes. So easy to learn, you'll be up and running in minutes, EDWin also features nice pricing, starting at just \$495.

Make your appointment with us today for the EDWin evaluation package. Welcome.

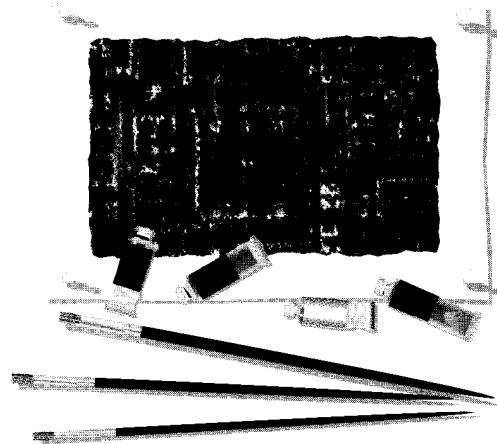
Vision EDA Corp.

995 E Baseline Rd. Ste 2166,
Tempe, Arizona 85283-1336

Phone: 1-800-EDA-4-YOU, or (602) 730 8900

Fax: (602) 730 8927

ELECTRONICS DESIGN FOR WINDOWS.



EDWIN

EDWin is a trademark of Norlinvest Ltd. Windows is a trademark of Microsoft Corp.