THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Adaptive Differential Pulse-Code-Modulation Coding

By J. R. BODDIE, J. D. JOHNSTON, C. A. McGONEGAL, J. W. UPTON, D. A. BERKLEY, R. E. CROCHIERE, and J. L. FLANAGAN

(Manuscript received June 26, 1980)

An adaptive differential pulse-code-modulation technique for encoding and decoding has been implemented using the Bell Laboratories digital signal processor integrated circuit. The encoder/decoder operates in real time and can accommodate 3- or 4-bit (8 kHz) encoding. In this paper, we discuss details of the implementation, the basic algorithm, and the features utilized in the digital signal processor.

I. INTRODUCTION

Adaptive differential pulse-code-modulation (ADPCM) encoding has been shown to be a simple and effective method for digitally encoding speech at bit rates in the range of approximately 24 to 48 kb/s.[1-6] At a rate of 24 kb/s, ADPCM can provide a good quality reproduction of speech that is acceptable for applications such as computer-controlled digital voice response systems.[4] At a rate of 32 kb/s, it can provide essentially a telephone bandwidth "transparent quality" (a quality that is indistinguishable from the original uncoded source) for a single tandem encoding. Adaptive differential pulse-code modulation has been studied for use in some types of transmission systems,[6] and for message storage and retrieval systems in which a reduction by a factor of two in bit rate over that of conventional 64 kb/s μ-law companded PCM is desired.[4]

Various forms of hardware have been suggested for the implementation of ADPCM coders. Some designs are based primarily on analog hardware[1,3] where parameters are pair-wise tuned between transmit-

ters and receivers. This results in problems with repeatability and stability in the analog designs. Bates[5] and Adelman, Ching, and Gotz[6] subsequently presented two different methods of designing all-digital ADPCM coders. The Bates approach uses a TTL logic design with a ROM-based look-up table for the adaptive step-size and an up/down counting scheme with digital adders and subtracters to avoid the use of a digital multiplier. The hardware requires a twelve-bit linear PCM input and has a total "package count" of approximately 80 standard TTL logic packages. It was constructed on two wire-wrapped Augat cards (one encoder and one decoder per board). The hardware by Adelman et al., was a ROM-based design that accepted a standard 64-kb/s μ-law companded PCM signal.

Recently, an LSI digital signal processor (DSP) has been developed by Bell Laboratories.[7] The DSP is a programmable processor capable of performing the entire ADPCM algorithm for multiple channels in a single LSI device. The ability of the processor to convert between conventional μ-255 companded PCM and two's complement binary code formats allows the ADPCM algorithm to use either format. The configuration of the ADPCM algorithm that is implemented on the DSP is described in Section II. Section III discusses details of how the algorithm is configured to use DSP features. Sections IV and V describe the hardware configuration and measured performance, respectively.

II. THE ADPCM ALGORITHM

2.1 Overall configuration

Figure 1 illustrates the basic configuration of the DSP implementation of ADPCM. The input analog signal $s(t)$ is sampled and A/D converted to an 8-bit μ-law companded PCM format to produce the sampled data signal $s(n)$, where n is the discrete time index. These operations are done externally to the DSP. Then $s(n)$ is converted in the DSP from μ-law format to a 20-bit linear PCM format for internal processing.

The ADPCM encoding is performed entirely within the transmitter DSP. The output is a 3- or 4-bit codeword, $I(n)$, which can be obtained through the normal output channel of the DSP. In the receiver, a

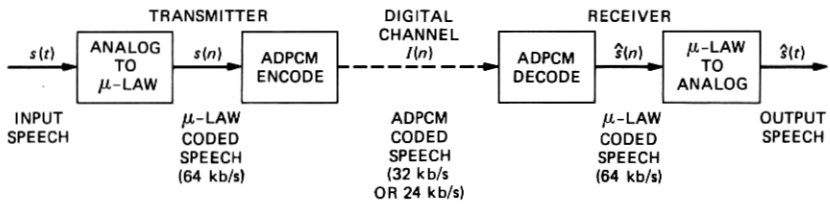

Fig. 1—Overall configuration of the ADPCM encoder/decoder using the DSP.

second DSP is used to decode $I(n)$ back to an 8-bit μ-law format. This signal is then converted to an analog signal with an external D/A converter. The clock and framing signals are carried between transmitter and receiver on separate lines. The transmission protocol for a given application depends on the data channel in which the system is implemented.

2.2 Block diagram of ADPCM

Figure 2 shows the block diagram of the ADPCM algorithm that was implemented on the DSP. The transmitter is shown in Fig. 2a and the receiver is shown in Fig. 2b. We assume that $s(n)$ in Fig. 2 is in a 20-bit linear PCM format, because of a direct 20-bit input to the DSP or because of an 8-bit μ-law to 20-bit linear PCM conversion that has been performed within the DSP.

The ADPCM algorithm can be partitioned into three basic parts (see Fig. 2): the adaptive PCM quantizer, the differential predictor loop, and the adaptive gain (step-size) control for the quantizer. We discuss these operations in the above order since this is the order in which they are computed.

2.3 Adaptive quantizer

A predicted value, $p(n)$, is first subtracted from the input signal $s(n)$ to form the difference signal

$$e(n) = s(n) - p(n). \tag{1}$$

The value of $p(n)$ is obtained from the predictor and is based on computations performed at the previous sample time, $n - 1$.

The difference signal $e(n)$ is then quantized by the adaptive quantizer to produce the ADPCM codeword $I(n)$. This adaptive quantization is achieved by first scaling $e(n)$ to the range of a 3- or 4-bit fixed step-size quantizer (see Fig. 2a). The scaled signal is denoted as

$$e_s(n) = \nabla(n) \cdot e(n), \tag{2}$$

where $\nabla(n)$ is an adaptive scale factor that is also determined from data available at the previous sample time $n - 1$. The combination of scaling by $\nabla(n)$ followed by the fixed step-size quantizer is equivalent to a quantizer with an adaptively varying step-size (which is inversely proportional to $\nabla(n)$).

Figure 3 shows the characteristics for a 3-bit (8 level) fixed step-size quantizer. The input signal $e_s(n)$ (consisting of an integer plus a fractional part) is converted to one of eight quantization levels in the range $-7/2$ to $+7/2$ corresponding respectively to integer codewords, $I(n)$, in the range -4 to $+3$. The output quantized signal is denoted as $\hat{e}_s(n)$ and is related to the code word $I(n)$ by the relation

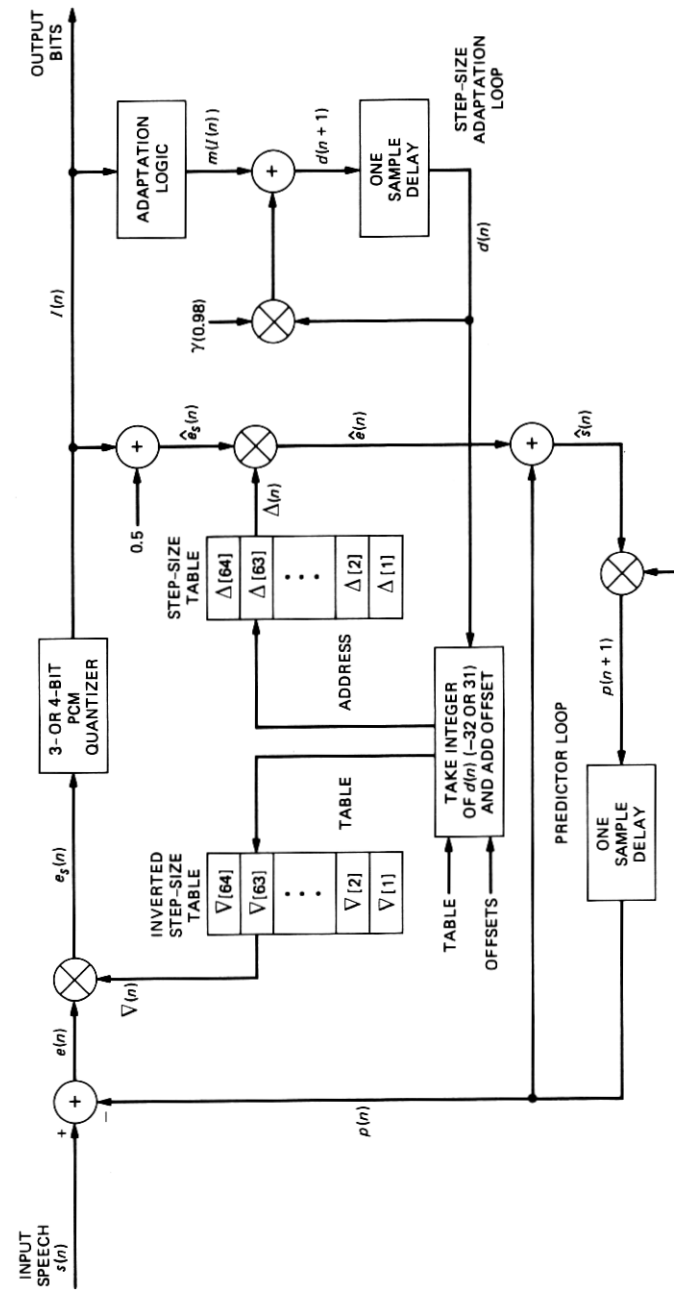

Fig. 2a—Block diagram of the ADPCM encoding algorithm.

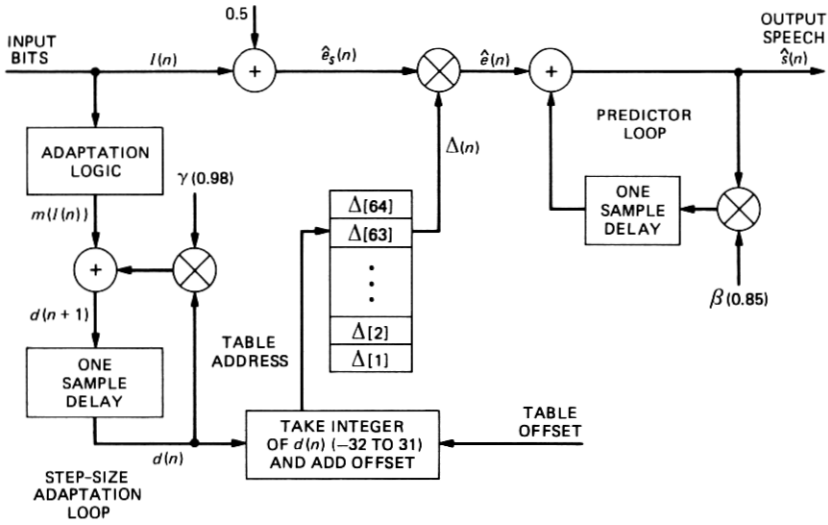

Fig. 2b—Block diagram of the ADPCM decoding algorithm.

$$\hat{e}_s(n) = I(n) + 0.5. \tag{3}$$

A similar quantizer characteristic is used for the 4-bit (16 level) design.

The unscaled, decoded difference signal can be obtained from $\hat{e}_s(n)$ by rescaling it by $\Delta(n)$, where $\Delta(n)$ is inversely related to $\nabla(n)$ and is directly proportional to the "step-size" of the equivalent adaptive quantizer. Thus,

$$\hat{e}(n) = \Delta(n) \cdot \hat{e}_s(n) \tag{4}$$

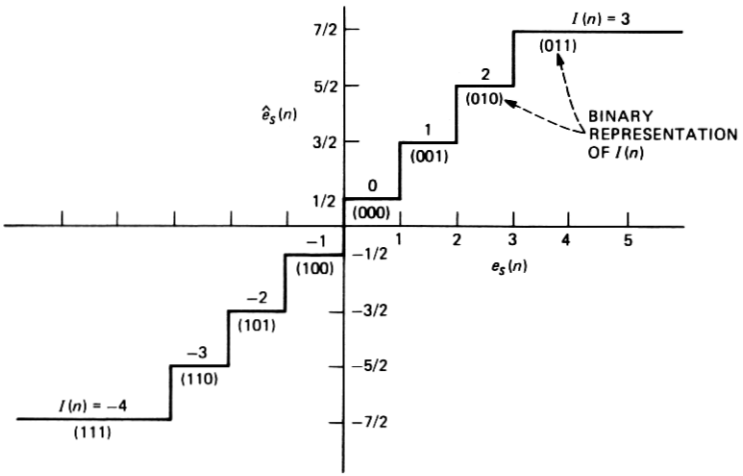

Fig. 3—Quantizer characteristic for the 3-bit PCM quantizer.

is the adaptively quantized representation of the difference signal $e(n)$. This completes the adaptive quantization part of the algorithm.

2.4 First-order predictor

The sum of $p(n)$ and $\hat{e}(n)$ also forms the adaptively quantized representation of the input signal (see Fig. 2a)

$$\hat{s}(n) = p(n) + \hat{e}(n). \tag{5}$$

However, the predictor value $p(n + 1)$ and the quantizer scale factors $\nabla(n + 1)$ and $\Delta(n + 1)$ need to be updated for the next sample time $n + 1$.

The new predictor value is computed as β times $\hat{s}(n)$; i.e.,

$$p(n + 1) = \beta \cdot \hat{s}(n), \tag{6}$$

where β is the predictor "leak" factor.[2] A value of $\beta = 0.85$ is often used for speech encoding. The value of $p(n + 1)$ is then stored for use at the next sample time $n + 1$.

2.5 Adaptive step-size control

The computation of the new value of $\Delta(n + 1)$ and $\nabla(n + 1)$ for the next time sample $n + 1$ is more involved. The algorithm used here is based on the robust step-size adaptation approach. The details and advantages of this form of the algorithm are discussed in considerable detail in Refs. 8 to 10. The method is as follows. The step-size referred to above is the effective spacing in the quantizer levels observed in the unscaled quantized signal $\hat{e}(n)$. Since this spacing is proportional to $\Delta(n)$, we will refer to $\Delta(n)$ in the following discussion as the step-size.

In the robust adaptation algorithm, the new step-size $\Delta(n + 1)$ is chosen as

$$\Delta(n + 1) = (\Delta(n))^{\gamma} \cdot M(I(n)). \tag{7}$$

That is, it is the old step-size $\Delta(n)$ raised to the power γ ($0 < \gamma \leq 1$, typically $\gamma = 0.98$) and scaled by a factor $M(\cdot)$ which is a function of the code word $I(n)$. If the code word is one of the upper magnitude levels of the quantizer [e.g., $I(n) = -4, -3, 2$, or 3 in Fig. 3], a value of $M(\cdot)$ greater than one is used to increase the step-size of the quantizer for the next sample time. If the codeword is one of the lower magnitude levels [e.g., $I(n) = -2, -1, 0, 1$ in Fig. 3], a value of $M(\cdot)$ less than one is used to reduce the quantizer step-size for the next sample time. In this way, the algorithm continually attempts to adapt the step-size such that the dynamic range of the adaptive quantizer is matched to the range of the signal $e(n)$.

A more direct approach for implementing this algorithm is obtained by expressing eq. (7) in logarithmic form. Let

$$d(n) = \log(\Delta(n)), \tag{8}$$

and

$$m(I(n)) = \log(M(I(n))), \tag{9}$$

where the base of the logarithm is determined by the choice of parameters in the coder, as discussed later. Then, eq. (7) becomes

$$d(n+1) = \gamma \cdot d(n) + m(I(n)), \tag{10}$$

and it has the form of a first-order difference equation. This is the equation that is implemented by the upper right-hand part of the block diagram in Fig. 2a.

The driving function $m(I(n))$ is a function of the code word $I(n)$, and it is determined in the adaptation logic algorithm which performs the function

$$m(I(n)) = \begin{cases} 8 \text{ if } |I(n) + 0.5| \geq 2.5, \\ -3 \text{ if } |I(n) + 0.5| < 2.5 \end{cases} \tag{11a}$$

for the 3-bit quantizer and

$$m(I(n)) = \begin{cases} 8 \text{ if } |I(n) + 0.5| \geq 4.5, \\ -3 \text{ if } |I(n) + 0.5| < 4.5 \end{cases} \tag{11b}$$

for the 4-bit quantizer.

The output of the step-size adaptation loop is the signal $d(n)$, which eq. (8) shows as the logarithm of the desired step-size.

Thus, to obtain $\Delta(n)$ (and $\nabla(n)$), we need to implement the relations

$$\Delta(n) = \exp(d(n)) \tag{12a}$$

and

$$\nabla(n) = \frac{1}{\Delta(n)} = \exp(-d(n)), \tag{12b}$$

where, again, the base of the logarithms and exponentials in eqs. (8) to (12) are determined by the choice of parameters of the coder, as discussed later.

The exponentials in eqs. (12a) and (12b) are computed by using look-up tables that are stored in the program ROM as indicated in Fig. 2a. The integer value of $d(n)$ is taken and constrained to the range $-32 \leq [d(n)] < 32$ for a lookup table size of 64. Table offset values are then added to this value to produce the appropriate ROM address locations. The tables are set up so that a value of $d(n) = 0$ points to the center of both tables ($\nabla[32]$ and $\Delta[32]$). Note that the bracketed values $\nabla[\cdot]$ and $\Delta[\cdot]$ in the tables of Fig. 2a refer to the contents of table locations addressed by $[\cdot]$ and not to sample times (which are referred to by the parenthesis notation $\nabla(n)$ and $\Delta(n)$).

The above algorithm uses 64 different step-size values of $\Delta(n)$ and

$\nabla(n)$. The table values are chosen to span the desired dynamic range of the input signal. If signals of larger or smaller amplitude than this are encountered, the step-size saturates at the upper or lower table values, respectively.

The table address locations are stored for use at the next sample time $n + 1$ to access values $\Delta(n + 1)$ and $\nabla(n + 1)$ as they are needed. This completes the operation of the ADPCM encoder.

2.6 Parameter selection for the step-size tables

The parameters of the step-size table were chosen such that the ratio of the maximum to minimum step-sizes in the table is 256, i.e.,

$$\Delta[64]/\Delta[1] = 256 = 2^8$$

and

$$\nabla[1]/\nabla[64] = 256.$$

This gives a step-size adaptation range of 48 dB (8 bits) that is appropriate for speech coding. Since there are 64 logarithmically spaced step-size values in each table, this corresponds to a 0.75-dB resolution, i.e., step-sizes increase by a ratio of 1.0902 in the table,

$$\Delta[i] = \Delta[1]\cdot(1.0902)^{i-1},$$

$$\nabla[i] = \nabla[1]\cdot(1.0902)^{-i+1}.$$

The maximum step-size is chosen so that the maximum range of $e(n)$ (approximately 17 bits) scales to the maximum quantizer range (3 or 4 bits). This prevents overloading on the high end of the dynamic range. Thus, for a 4-bit quantizer

$$\Delta(n)\,|_{\max} = \Delta[64] = 2^{17-4} = 2^{13}$$

and

$$\Delta(n)\,|_{\min} = \Delta[1] = 2^5.$$

For a 3-bit quantizer, values of $\Delta[\cdot]$ should be increased by two and values of $\nabla[\cdot]$ should be decreased by two for best dynamic range performance.

The manner in which these table values are scaled and stored in the DSP will be explained more fully in Section 3.4.

2.7 Decoder for ADPCM

Figure 2b shows a similar block diagram for the ADPCM decoder. The input code word $I(n)$ is converted to the decoded difference signal $\hat{e}(n)$ by adding 0.5 (see Fig. 3) and scaling the result by $\Delta(n)$. The decoded output signal $\hat{s}(n)$ is then obtained by accumulating values $\hat{e}(n)$ in the predictor loop in the same manner as in the encoder. The new step-

size is then computed in an exactly duplicate manner to that in the encoder. Thus, the ADPCM decoder duplicates a subset of the encoder block diagram.

III. PROGRAMMING TECHNIQUES UTILIZED IN THE DSP FOR THE ADPCM ALGORITHM

The ADPCM algorithm, as it is configured above, is designed so that it can be conveniently implemented on the DSP. We have already alluded to ways in which the DSP is used in implementing the algorithm. In this section, we point out some additional aspects of the program and discuss how some of the unique features of the DSP are used. Discussions in this section use the 4-bit algorithm as an example.

3.1 Memory utilization

The encoder program and the step-size tables for $\Delta[\cdot]$ and $\nabla[\cdot]$ are stored in ROM occupying approximately 170 words of memory. Five RAM locations are used for storing the following values: $2(I(n) + 0.5)$, $2\,\hat{s}(n)$, $p(n)$, $d(n)$ and the integer version of $d(n)$. Access to the step-size tables is made by setting the RX register in the DSP to the center address of the desired table. The table address for the appropriate step-size is then obtained by setting the K register to the integer value $[d(n)]$ (limited to the range -32 to 31) and then incrementing RX by the value K using the rxk command. The RX register then contains the ROM address for the appropriate step-size $\nabla[\cdot]$. The DSP instructions that implement this technique are as follows:

```
rya = 5;                "RAM pointer to int [d(n)]"
 rx = &TABLE;           "set rx pointer to ∇[32]"
  k = rym;;;            "set k to int [d(n)]"
  a = p p = rxk*c;      "pointer to appropriate ∇[·]"
```

Note that this technique only works for table sizes up to 256, since the K register is limited to 8 bits.

3.2 Quantization

The computation of the quantizer input, $e_s(n)$, is accomplished in a straightforward manner on the DSP by using a subtraction and a multiplication. The conversion from $e_s(n)$ to a 4-bit integer $I(n)$ (according to the quantizer characteristic in Fig. 3) is done by truncating the fractional part of $e_s(n)$.

Assuming $e(n)$ has been computed and stored in the W register and RX is pointing to the appropriate step size, the following DSP instructions compute $I'(n)$.

```
                    p = 1*c;
a = p               p = rxj*w;      "compute e_s(n) = ∇(n)·e(n)"
a = p + a/2         p = −1*c;       "compute e_s(n) + 0.5"
a = p & a;                          "truncate to form I'(n)"
```

The truncation is achieved by using the accumulator control statement a = p & a, which performs a bit-by-bit AND operation between the P and A registers in the DSP. $I'(n)$ is stored in the A register and the (binary) number 1111 ··· 111.000 ··· is stored in the P register by the instruction p = −1*c (c = 2^{14}). The operation zeros out the fractional part of $(e_s(n) + 0.5)$ and leaves the integer part untouched. The resulting integer $I'(n)$ is then constrained to the range $-8 \leq I'(n) \leq 7$ (for 4-bit quantization) to form the desired codeword $I(n)$. This is done using the conditional AU operation as shown in the following instructions.

```
                    p = 8*c;
    a = p + a       p = 0*c;
if (a < 0) doau ();  a = p;         "if (I'(n) < −8) I'(n) = − 8"
    a = p + a       p = −15*c;
    a = p + a       p = 0*c;
if (a > 0) doau ();  a = p;         "if (I'(n) > 7) I'(n) = 7"
    a = p + a       p = 7*c;
    a = p + a                       "4-bit I(n)"
```

This yields the 4-bit value $I(n)$ in the accumulator.

3.3 Internal scaling of data

At the output of the quantizer a value of 0.5 is added to $I(n)$ (see Fig. 2) to produce $\hat{e}_s(n)$, which is then multipled by $\Delta(n)$. To accomplish this, $\hat{e}_s(n)$ must be transferred from the A register to the W register and the input of the multiplier. Since this transfer from the A to W registers truncates the fractional part of the value in the A register, it is first scaled by a factor of 2 to avoid the truncation of the fractional part. This scale factor of two is carried through the computation of $\hat{s}(n)$ in the block diagram of Fig. 2.

3.4 Scaling of step-size table values

The values stored in the step-size tables in ROM are more conveniently handled if they are scaled to be less than 2 in magnitude. The internal DSP arithmetic is such that 16-bit values from ROM are assumed to have 14 fractional bits. This is appropriate for the inverse step-size table $\nabla[\cdot]$ in which numbers $(e(n))$ in a 17-bit range are scaled down to a 4-bit range $(e_s(n))$. This table takes on values:

$$\begin{aligned}\nabla[1] &= 0.031250\\ \nabla[2] &= 0.028617\\ &\vdots\\ \nabla[63] &= 0.000133\\ \nabla[64] &= 0.000122\end{aligned}$$

which can be directly stored in the ROM.

In the step-size table $\Delta[\cdot]$, the inverse of these numbers must be stored. To accomplish this, the inverses are first scaled by a factor of 2^{-14} and, thus, they take on the fractional range (denoted by primes):

$$\begin{aligned}\Delta'[1] &= 0.001953\\ \Delta'[2] &= 0.002133\\ &\vdots\\ \Delta'[63] &= 0.457650\\ \Delta'[64] &= 0.499754\end{aligned}$$

When these table values are used in the multiplication, the resulting product $\Delta'(n)\hat{e}_s(n)$ is scaled back up by a factor 2^{14} using the 14-bit shift option $a = a \ll 14$.

The inverse relationship between the table values requires that

$$\nabla[i]\cdot\Delta'[i]\cdot 2^{14} = 1,$$
$$i = 1, 2, \cdots, 64. \tag{14}$$

Since $\nabla[i]$ and $\Delta'[i]$ must be quantized to 16-bit numbers (14-bit fractions), for storage in the ROM, the condition in eq. (14) cannot be met exactly. To obtain the greatest accuracy in representing these numbers, the smaller of the two values $\nabla[i]$ or $\Delta'[i]$ for each value of i is quantized first. The reciprocal of this number is then computed (with floating point accuracy) and scaled by 2^{-14}. This value is then quantized to the 16-bit range of the ROM to produce the inverse table value. Thus, the accuracy of the condition in eq. (14) is maintained as closely as possible.

3.5 Control of the address range for the tables

Another unique part of the program involves the control of the range of the table address pointer $d(n)$ to the range -32 to 31 (excluding the table offset). This is accomplished with the aid of the overflow protection feature of the DSP when data is transferred from the A register to the W register. The operations in the step-size control loop are scaled so that $d(n)$ is computed in the upper range of the A register. When this number is transferred to the W register, it is automatically limited to the proper range by the overflow protection option in the DSP. Scaling this number back down to a 6-bit integer range gives the

desired range of −32 to 31 for the table location (excluding the table offset).

Specifically, this is accomplished as follows. First the driving function

$$m(I(n)) = \begin{cases} 8 \cdot 2^8 \text{ if } |2(I(n) + 0.5)| \geq 9, \\ -3 \cdot 2^8 \text{ if } |2(I(n) + 0.5)| < 9 \end{cases}$$

is computed and stored in the A register. The value $m(I(n))$ is then multiplied by 8 twice and added to $\gamma \cdot d(n)$ to form $d(n + 1)$ scaled by 2^{14}. The value $d(n + 1)$ is limited when it is transferred to the W register, and the table look-up offset, int $(d(n))$, is obtained by multiplying the W register by 2^{10}. This result is directly loaded into the K register as discussed in Section 3.1. Assuming the A register contains the absolute value of 2 $(I(n) + 0.5) - 9$, and the RY register points to $d(n)$, the following DSP instructions compute $d(n + 1)$ and the table offset value, integer $d(n)$.

```
                                    p = 04000*c;
                  if (a < 0) doau ();                  "if (a ≥ 0) m = 8*2^8"
                  a = p             p = 0176400*c;     "else m = −3*2^8"
                  a = p             p = 0*c;
                  a = p + 8*a       p = .98*rym;       "compute γ*d(n)"
                  a = p + 8*a       p = 0*c;           "compute d(n + 1)*2^14"
          w = a;                                       "overflow protection"
rdp = w           a = p             p = 16*w;          "save d(n + 1)"
                  a = p;
          w = a;
rdp = w;                                               "save int(d(n))"
```

IV. HARDWARE CONFIGURATION

The hardware used to implement the algorithm described in this paper consists of two 16.5- by 11.5-cm wire wrap cards, one for the transmitter and one for the receiver. Both cards are of identical construction with jumper plugs used to determine whether the card will act as a transmitter or a receiver in the configuration shown in Fig. 1.

Figure 4 shows a more detailed block diagram of the cards. Each card contains a DSP, along with a 2048-word by 16-bit PROM holding the program and step-size tables. (For permanent applications the ROM would be integral to the DSP). In addition, there are analog filters, μ-law encoder and decoder chips, clock generators, and synchronization logic.

The ADPCM transmitter card accepts an analog input which is applied to a buffer amplifier and then to a low-pass filter and a μ-law encoder. The sampling rate of this encoder is determined by the repetition period of the sync signal produced by the sync generator. The serial output of the encoder is shifted out by the clock signal. These three

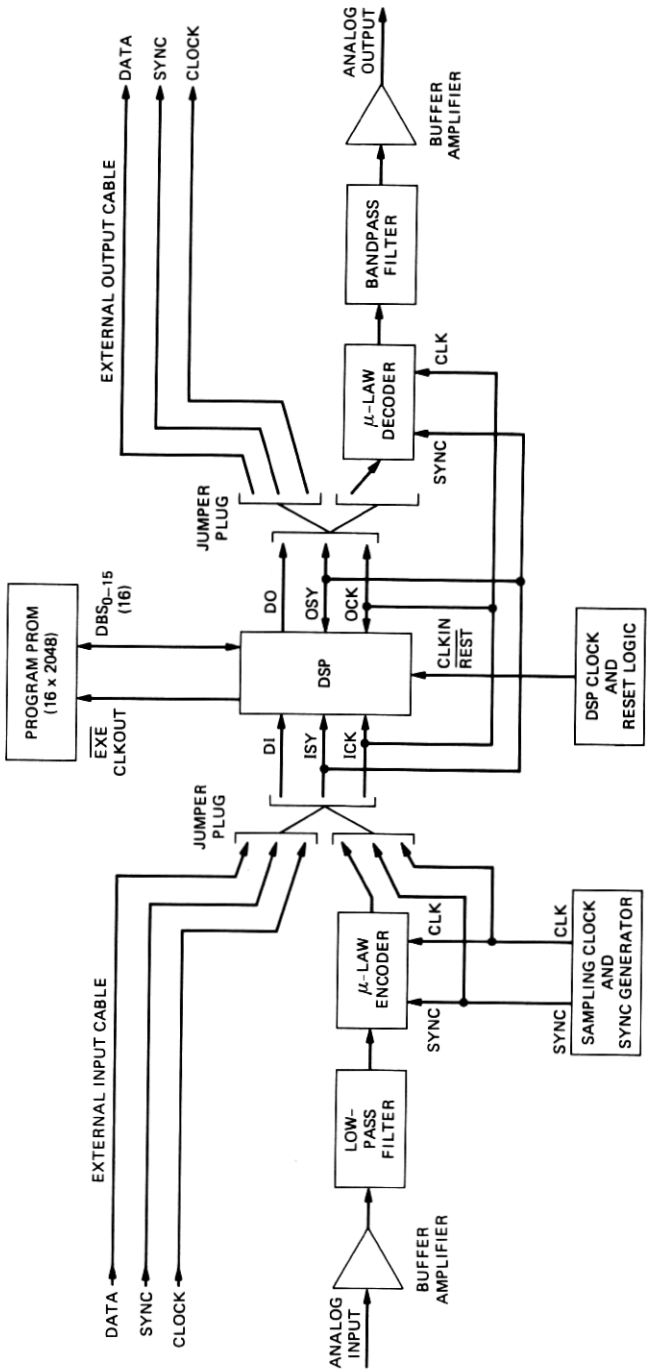

Fig. 4—Hardware configuration for DSP.

signals (data, sync, and clock) are connected through a jumper plug to the data in (DI), input sync (ISY), and input clock (ICK) pins of the DSP. Operating from the encoding program stored in the PROM, the DSP converts the incoming data into ADPCM code words which become available at the data output pin (DO). The shifting of this word out of the DSP is controlled by output sync (OSY) and output clock (OCK) lines. These three signals are connected through another jumper plug to the external output cable which sends them to the receiver card. The μ-law decoder on the transmitter card is not used.

The ADPCM receiver card receives the data, sync, and the clock signals from the transmitter card over the external input cable, which is connected through the first jumper plug directly into the DSP. The μ-law encoder and sync generator on this card are bypassed. The DSP takes the incoming ADPCM code words and converts them to μ-law PCM, according to the instructions of the decoding algorithm in its PROM. The μ-law data words are shifted out of the DSP on the DO line and are connected through the other jumper plug to the input of the μ-law decoder chip. The analog output of the decoder is bandpass filtered and then sent through a buffer amplifier to produce the reconstructed analog output signal.

The cards used here are not limited to only one application of the DSP. By setting the jumper plugs so that audio input and output are both on the same card, different types of filtering algorithms can be tested. Conversely, both the input and output of a card can be connected over the external cables. In this manner, several DSPs can be connected together serially for more complex signal processing.

V. PERFORMANCE

Figure 5 shows the signal-to-noise ratio measured for the 4-bit ADPCM/μ-law coder (Fig. 1) as a function of frequency for input signal

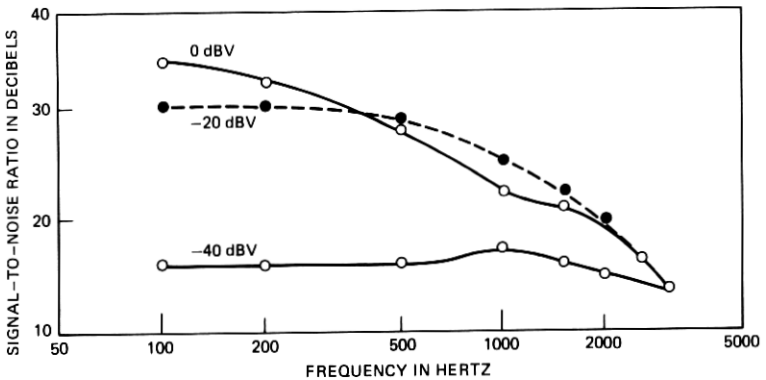

Fig. 5—Signal-to-noise ratio performance of the 4-bit (32 kb/s) ADPCM/μ-law coding configuration of Fig. 1.

levels of 0, −20, and −40 dB of full scale. It corresponds closely to that of the coders in Ref. 3. The dynamic range is limited to about 48 dB because of the range of the step-size tables and the μ-law encoders. This dynamic range performance can be modified (within the limitations of the μ-law encoders) by changing the step-size tables. At low input levels the significant noise is that of the μ-law encoders.

The subjective quality of the 4-bit ADPCM is very similar to 200- to 3200-Hz (telephone band) filtered speech without encoding. This suggests that a single ADPCM link can provide essentially a "transparent" quality for telephone bandwidth speech.

VI. CONCLUSIONS

In this paper, we have discussed the implementation of the ADPCM algorithm on the DSP. The encoder program uses 26 percent of the 8-kHz real-time capability of the DSP running with a 5-MHz clock. It uses 4 percent of the RAM and 17 percent of the ROM. The decoder program uses 22 percent of the real-time capabilities, 4 percent of the RAM and 10 percent of the ROM. This suggests that 4 encoders, 4 decoders, or 2 encoder-decoders could be implemented on a single DSP. The program uses a number of unique features of the DSP to achieve an efficient implementation of the algorithm, and it demonstrates the flexibility of the DSP in doing small-to-medium scale algorithms of this type.

REFERENCES

1. P. Cummiskey, N. S. Jayant, and J. L. Flanagan, "Adaptive Quantization in Differential PCM Coding of Speech," B.S.T.J., *52,* No. 7 (September 1973), pp. 1105–18.
2. N. S. Jayant, "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," Proc. IEEE, *62* (May 1974), pp. 611–32.
3. J. D. Johnston and D. J. Goodman, "Multipurpose Hardware for Digital Coding of Audio Signals," IEEE Trans. Commun., *COM-26,* No. 11 (November 1978), pp. 1785–8.
4. L. H. Rosenthal et al., "A Multiline Computer Voice Response System Utilizing ADPCM Coded Speech," IEEE Trans. Acoust., Speech, Sig. Proc., *ASSP-22,* No. 5 (October 1974), pp. 339–52.
5. S. Bates, "A Hardware Realization of a PCM-ADPCM Code Converter," S. M. Dissertation, Dept. of Electrical Engineering, Massachusetts Institute of Technology, 1976.
6. H. W. Adelmann, Y. C. Ching, and B. Gotz, "An ADPCM Approach to Reduce the Bit Rate of μ-Law Encoded Speech," B.S.T.J., *58,* No. 7 (September 1979), pp. 1659–71.
7. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
8. D. J. Goodman and R. M. Wilkinson, "A Robust Adaptive Quantizer," IEEE Trans. Commun., *COM-23* (November 1975), pp. 1362–5.
9. D. Mitra, "An Almost Linear Relationship Between the Step-Size Behavior and the Input Signal Intensity in Robust Adaptive Quantization," IEEE Trans. Commun., *COM-27* (March 1979), pp. 623–9.
10. J. D. Johnston and R. E. Crochiere, "An All Digital "Commentary Grade" Subband Coder," J. Audio Eng. Soc., *27,* No. 11 (November 1979), pp. 855–65.