Application Note

# Implementation of Adaptive Differential Pulse-Code Modulation (ADPCM) Transcoder with the *WE*® DSP16 Digital Signal Processor

## Contents



*Contributed by:* *Y. Lai*

## Introduction

Direct access to the digital channel is now provided within some private branch exchange (PBX) environments, in private digital networks, by digital data services (DDS), and will be provided through integrated services digital network (ISDN) connections. To make better use of these digital channels, more sophisticated techniques can be applied to coding analog signals at a rate lower than 64 kbits/s so that more than one analog signal, or concurrent analog and digital signals, can be transmitted over the same 64 kbits/s channel.

Typically, telephone quality signals are digitally encoded using pulse-code modulation (PCM) by sampling the analog signal at an 8 kHz rate, and quantizing each sample using an 8-bit companding codec. This conversion produces a 64 kbits/s digital signal from the original 4 kHz bandwidth analog signal. The digital signal is then transmitted through the telephone network using a 64 kbits/s channel before being converted back to the analog domain at the receiving end. One solution for increasing the capacity of the 64 kbits/s channels is to encode the voice channel using ADPCM.

The ADPCM transcoder converts an A-law or μ-law 64 kbits/s PCM data stream into a 32 kbits/s ADPCM data stream. Once converted, the channel capacity of the current 64 kbits/s PCM is doubled, providing more channels for transmitting and receiving data. A block diagram of a full-duplex ADPCM transcoder is shown in Figure 1.

Two programs for implementing the ADPCM transcoder are described in this note. The programs implement a transcoder meeting the

CCITT recommendation G.721 [1]*. A full-duplex implementation requires a DSP16A running at 50 ns. This set of programs can be modified to allow nonstandard operation on a DSP16 [2] running at 75 ns. Equivalent coding quality can be achieved in this mode. This should be the program of choice when interoperation with a CCITT G.721 transcoder is not required. The modification is identified in the Source Code section.

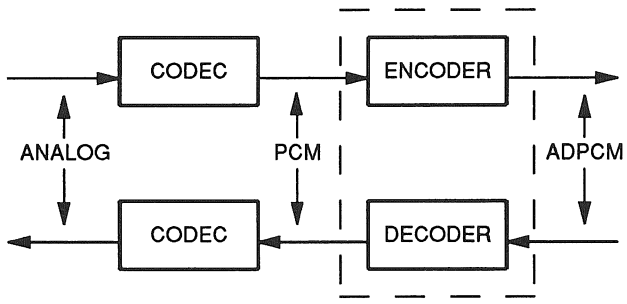



**Figure 1. Full-Duplex ADPCM Transcoder**

# The ADPCM Algorithm

The bit rate needed to transmit (or store) speech can be reduced by removing any redundancy in the original PCM encoded signal. ADPCM transcoders remove redundancy by using a predictor to estimate the signal value at each sample. By subtracting this estimate from the original signal, a signal with less dynamic range is produced, which, in turn can be quantized with fewer bits than the original PCM sample. Adaptation of the predictor is based on a history of recent quantized samples. Therefore, no additional information is needed by the decoder to reconstruct the transmitted signal. The quantizer step-size adaptation rate is also controlled to optimize the transcoders signal-to-noise ratio. The quantizer stops adapting in the presence of narrow-band energy to improve the performance for voiceband data signals. The decoder also includes a synchronous coding adjuster which reduces the cumulative distortion that occurs on synchronous tandem encodings.

The detailed functional blocks comprising the transcoder are described in the Encoder and Decoder sections.

## Encoder

*Input Conversion and Error Calculation*

A detailed diagram of the ADPCM encoder is shown in Figure 2. The input signal, s(k), is a 64 kbits/s μ-law PCM or A-law data stream. It is converted to a linear PCM signal, $s_l(k)$, by the PCM format converter. The difference signal, d(k), is calculated from the linear PCM signal and the signal estimate, as shown by the formula in Equation 1.

1) $d(k) = s_l(k) - s_e(k)$

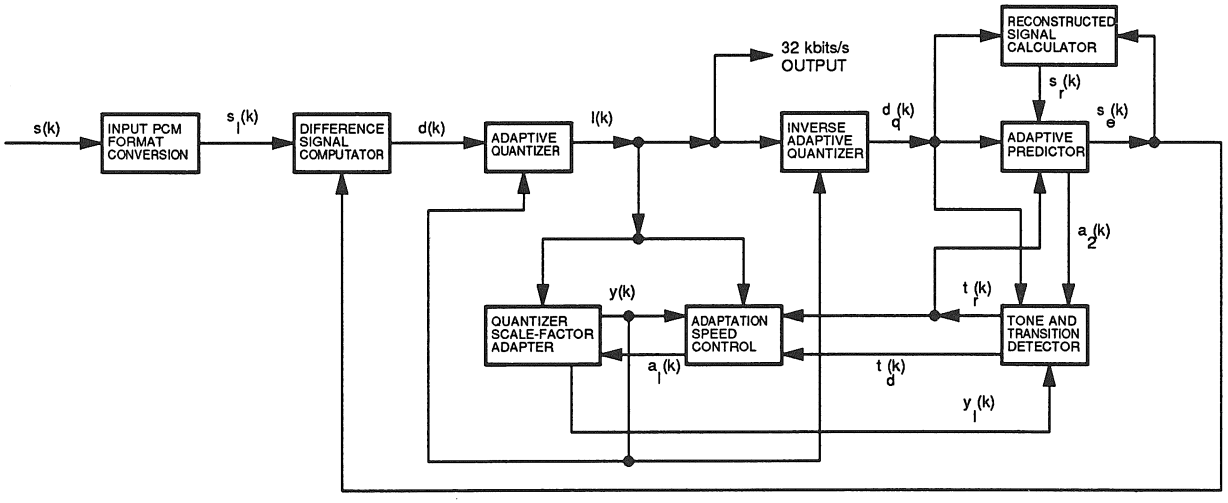



**Figure 2. ADPCM Encoder Block Diagram**

* [ ] indicates a reference listed at the conclusion of this application note.

*Adaptive Quantizer*

The difference signal is then fed to a 15-level adaptive quantizer from which the 4-bit quantized output signal, I(k), is generated. Prior to quantization of the signal d(k), it is converted to a base 2 logarithmic representation and scaled by y(k), which is computed by the quantizer scale-factor adaptation block. The normalized input/output characteristics of the quantizer for the 32 kbits/s operation are given in Table 1. I(k) is the 32 kbits/s ADPCM encoded output signal transmitted to the receiver and fed back into the transmitter.

**Table 1. Quantizer Normalized Input/Output Characteristic for 32 kbits/s Operation**

| Normalized Quantizer Input Range $\log_2 \lvert d(k)\rvert - y(k)$ | $\lvert I(k)\rvert$ | Normalized Quantizer Output $\log_2 \lvert d_q(k)\rvert - y(k)$ |
|---|---|---|
| [3.12, + ∞) | 7 | 3.32 |
| [2.72, 3.12) | 6 | 2.91 |
| [2.34, 2.72) | 5 | 2.52 |
| [1.91, 2.34) | 4 | 2.13 |
| [1.38, 1.91) | 3 | 1.66 |
| [0.62, 1.38) | 2 | 1.05 |
| [–0.98, 0.62) | 1 | 0.031 |
| (– ∞, –0.98) | 0 | – ∞ |

Note: [ or ] indicates that the endpoint value is included in the range and ( or ) indicates that the endpoint value is excluded from the range.

*Inverse Adaptive Quantizer*

The inverse adaptive quantizer generates the quantized difference signal, $d_q(k)$, by scaling specific values of y(k) from the appropriate normalized quantizing characteristic, shown in Table 1, and converting the result to linear form.

*Quantizer Scale-Factor Adapter*

The quantizer scale-factor adapter computes the scaling factor, y(k), which determines the quantizer step size. The scaling factor is a combination of two terms, the unlocked scaling factor, $y_u(k)$, and the locked scaling factor, $y_l(k)$. The scaling factors can be calculated using the following formulas:

2a) $y(k) = a_l(k)y_u(k-1) + [1-a_l(k)]y_l(k-1)$

where $0 \le a_l(k) \le 1$

2b) $y_u(k) = (1-2^{-5})y(k) + 2^{-5}W[I(k)]$

where $y_u(k)$ is limited by $1.06 \le y_u(k) \le 10.00$.

2c) $y_l(k) = (1-2^{-6})y_l(k-1) + 2^{-6}y_u(k)$

The definition of the signal W[I(k)] is shown in Table 2.

**Table 2. Definition of W[I(k)] for 32 kbits/s Encoding**

| $\lvert I(k)\rvert$ | 7 | 6 | 5 | 4 |
|---|---|---|---|---|
| W[I(k)] | 70.13 | 22.19 | 12.38 | 7.00 |
| $\lvert I(k)\rvert$ | 3 | 2 | 1 | 0 |
| W[I(k)] | 4.00 | 2.56 | 1.13 | –0.75 |

*Adaptation Speed Control*

The adaptation speed control allows two modes of operation, fast and slow. The two modes provide better performance in signals with large (e.g., speech) and small (e.g., data) variance, respectively. The adaptation speed-control parameter, $a_l(k)$, is derived from a measure of the rate of change of the difference signal values.

Two measures of the average magnitude of I(k) are computed by using the following equations:

3a) $d_{ms}(k) = (1-2^{-5})d_{ms}(k-1) + 2^{-5}F[I(k)]$

and

3b) $d_{ml}(k) = (1-2^{-7})d_{ml}(k-1) + 2^{-7}F[I(k)]$

The definition of the signal F[I(k)] is shown in Table 3.

**Table 3. Definition of F[I(k)] for 32 kbits/s Encoding**

| $\lvert I(k)\rvert$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| F[I(k)] | 7 | 3 | 1 | 1 | 1 | 0 | 0 | 0 |

Therefore, $d_{ms}(k)$ is a relatively short-term average of F[I(k)], and $d_{ml}(k)$ is a relatively long-term average of F[I(k)]. These two averages can be used to define $a_p(k)$.

$$4a)\quad a_p(k) = \begin{cases} (1-2^{-4})a_p(k-1) + 2^{-3}, \\ \quad \text{if } |\, d_{ms}(k) - d_{ml}(k) \,| \geq 2^{-3} d_{ml}(k) \\ (1-2^{-4})a_p(k-1) + 2^{-3}, \\ \quad \text{if } y(k) < 3 \\ (1-2^{-4})a_p(k-1) + 2^{-3}, \\ \quad \text{if } t_d(k) = 1 \\ 1, \text{ if } t_r(k) = 1 \\ (1-2^{-4})a_p(k-1), \\ \quad \text{otherwise.} \end{cases}$$

The equation $a_p(k-1)$ is limited to yield $a_l(k)$.

$$4b)\quad a_l(k) = \begin{cases} 1, & a_p(k-1) > 1 \\ a_p(k-1), & a_p(k-1) \leq 1 \end{cases}$$

The signal $d_q(k)$ is then fed to the reconstructed signal calculator, an adaptive predictor, and a tone and transition detector.

*Adaptive Predictor and Resconstructed Signal Calculator*

The adaptive predictor and the reconstructed signal calculator compute the signal estimate, $s_e(k)$, by using the following formulas:

$$5a)\quad s_e(k) = \sum_{i=1}^{2} a_i(k-1)s_r(k-i) + s_{ez}(k)$$

$$5b)\quad s_{ez}(k) = \sum_{i=1}^{6} b_i(k-1)d_q(k-i)$$

$$5c)\quad s_r(k-i) = s_e(k-i) + d_q(k-i)$$

$$5d)\quad a_1(k) = [1-2^{-8}]\, a_1(k-1) + 3*2^{-8}\text{sgn}\,[p(k)]\,\text{sgn}[p(k-1)]$$

$$a_2(k) = [1-2^{-7}]\, a_2(k-1) + 2^{-7}\{\text{sgn}\,[p(k)]\,\text{sgn}[p(k-2)] - f[a_1(k-1)]\,\text{sgn}\,[p(k)]\,\text{sgn}\,[p(k-1)]\}$$

where

$$p(k) = d_q(k) + s_{ez}(k)$$

$$f(a_1) = \begin{cases} 4a_1, & |a_1| \leq 2^{-1} \\ 2\text{sgn}(a_1), & |a_1| > 2^{-1} \end{cases}$$

and sgn[0] = 1, except sgn[p(k–i)] is defined to be 0 only if p(k–i) = 0 and i = 0; with the stability constraints:

$$|a_2(k)| \leq 0.75 \text{ and } |a_1(k)| \leq 1 - 2^{-4} - a_2(k)$$

If $t_r(k) = 1$, then $a_1(k) = a_2(k) = 0$

$$5e)\quad b_i(k) = [1-2^{-8}]\, b_i(k-i) + 2^{-7}\text{sgn}\,[d_q(k)]\,\text{sgn}[d_q(k-i)]$$

where $i = 1,2,...6$ and $-2 \leq b_i(k) \leq +2$

and sgn[0] = 1, except $\text{sgn}[d_q(k-i)]$ is defined to be 0 only if $d_q(k-i) = 0$ and i=0. If $t_r(k) = 1$, then $b_1(k) = b_2(k) = \cdots = b_6(k) = 0$.

*Tone and Transition Detector*

The tone and transition detector is used to improve performance of modems transmitting through an ADPCM encoder. This is accomplished by controlling the adaptation rate of the quantizer step size and the predictor coefficient values. A two-step detection process is used to accomplish this control. First, if the presence of a partial band signal (e.g., tone) is detected, the quantizer is set to adapt at a fast rate.

$$6a)\quad t_d(k) = \begin{cases} 1, & a_2(k) < -0.71875 \\ 0, & \text{otherwise} \end{cases}$$

If a transition from a partial band signal to a wide band signal (e.g. speech) is detected, the predictor coefficients are set to zero, and the quantizer is set to a fast mode of adaptation.

$$6b)\quad t_r(k) = \begin{cases} 1, & a_2(k) < -0.71875 \text{ and} \\ & |d_q(k)| > 24 * 2^{y_l(k)} \\ 0, & \text{otherwise} \end{cases}$$

## Decoder

A detailed diagram of the ADPCM decoder is shown in Figure 3. The input signal, I(k), is a 32 kbits/s ADPCM signal. Most of the functional blocks comprising the decoder are described in the encoder section, with one exception. The output of the PCM format converter is fed to a synchronous coding adjuster. The function of the synchronous coding adjuster is to prevent cumulative distortion that occurs on synchronous tandem codings. The synchronous coding adjuster, $s_d(k)$, produces an output which is close to the original signal that was processed by the ADPCM encoder. This is accomplished by considering the values of $d_x(k)$, (a version of the encoded signal), and $s_e(k)$, (the predicted signal error), as follows:

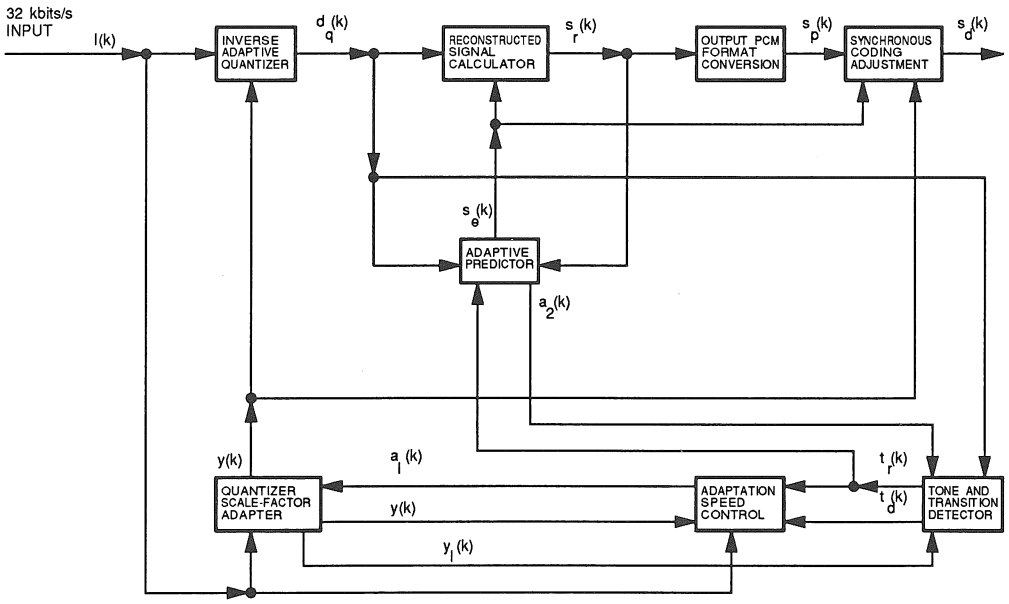



**Figure 3. ADPCM Decoder Block Diagram**

$$7a)\quad s_d(k) = \begin{cases} s_p^+(k), & d_x(k) < \text{lower interval boundary} \\ s_p^-(k), & d_x(k) \geq \text{upper interval boundary} \\ s_p(k), & \text{otherwise} \end{cases}$$

where

$s_p(k)$ = the output PCM code word of the decoder.

$s_p^+(k)$ = the PCM code word that represents the next more positive PCM output level [when $s_p(k)$ represents the most positive output level, then $s_p^+(k)$ is constrained to be the value $s_p(k)$].

$s_p^-(k)$ = the PCM code word that represents the next more negative PCM output level [when $s_p(k)$ represents the most negative output level, then $s_p^-(k)$ is constrained to be the value $s_p(k)$].

$$7b)\quad d_x(k) = s_{lx}(k) - s_e(k)$$

where $s_{lx}(k)$ is a linear PCM version of the signal $s_p(k)$.

# Implementation

This section describes the hardware and software aspects of a CCITT ADPCM transcoder implementation using the DSP16. Alternative code for the more efficient, nonstandard implementation is also provided in the source code listing.

## Hardware Interface

Figure 4 is a schematic of hardware that can be used to demonstrate both the CCITT and non-standard ADPCM algorithms. The hardware consists of two identical units comprised of an AT&T T7500 Codec [3] interfaced to a DSP16 through its serial port. A common clock is used to provide codec interface signals for both units. The 32 kbits/s ADPCM signal is transmitted between the two DSP16s using the parallel port (PIO). The port is configured in status/control (S/C) mode. It is clocked at 32 kHz through the parallel input data strobe (PIDS) by a divided version of the codec bit clock. The S/C mode allows data to be simultaneously transferred into the upper 8 bits of the 16-bit PIO and out of its lower 8 bits. In this case, only one bit in each direction is needed to transmit a full-duplex, 32 kbits/s bit stream.

Figures 5 through 7 show the interface timing for the DSP16 serial port and the codec. Figure 8 shows the timing for the DSP16 PIO. The parallel output data strobe (PODS) is tied low in both DSP16s to force the lower 8 bits of the PIO into output mode.

When implementing the CCITT algorithm, DSP16 #1 is used to encode the analog signal on the input of its corresponding codec. The 32 kbits/s encoded signal is transmitted to DSP16 #2 that decodes the signal and outputs the analog signal through its corresponding codec. In the nonstandard mode,

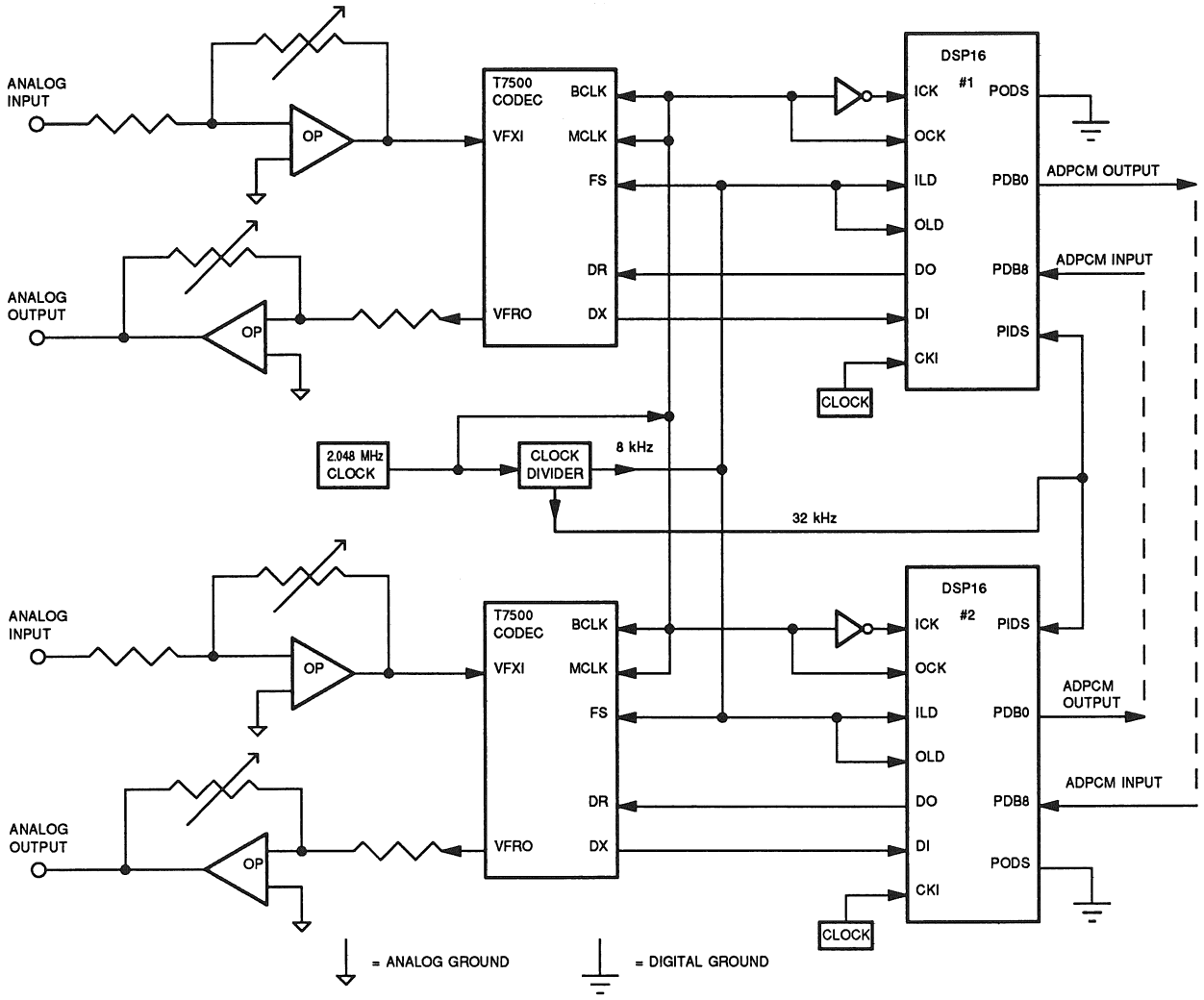
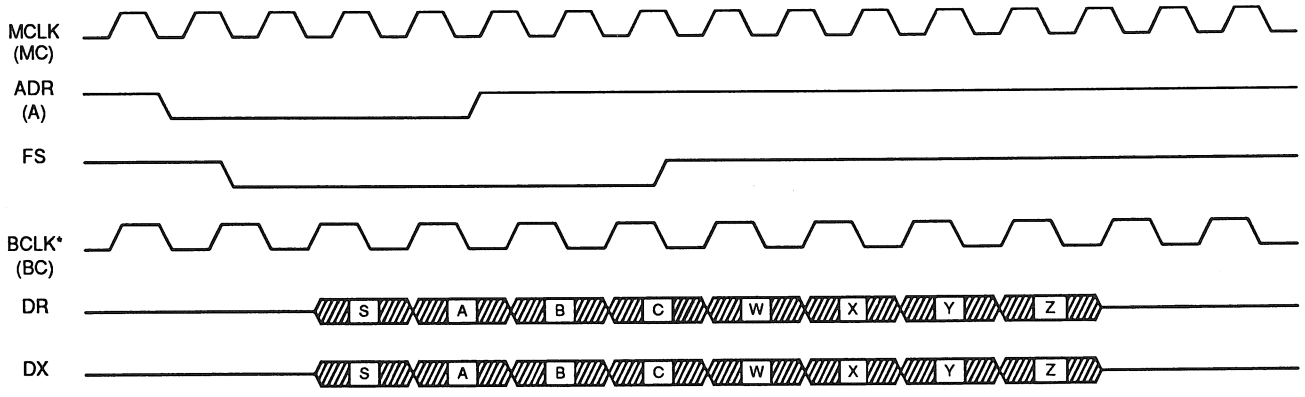



**Figure 4. Standard ADPCM Mode Implementation Example**




*BCLK ranges from 128 kHz to 4.096 MHz, it determines ADR, FS, DR, and DX timing

**Figure 5. T7500 Codec I/O Timing**

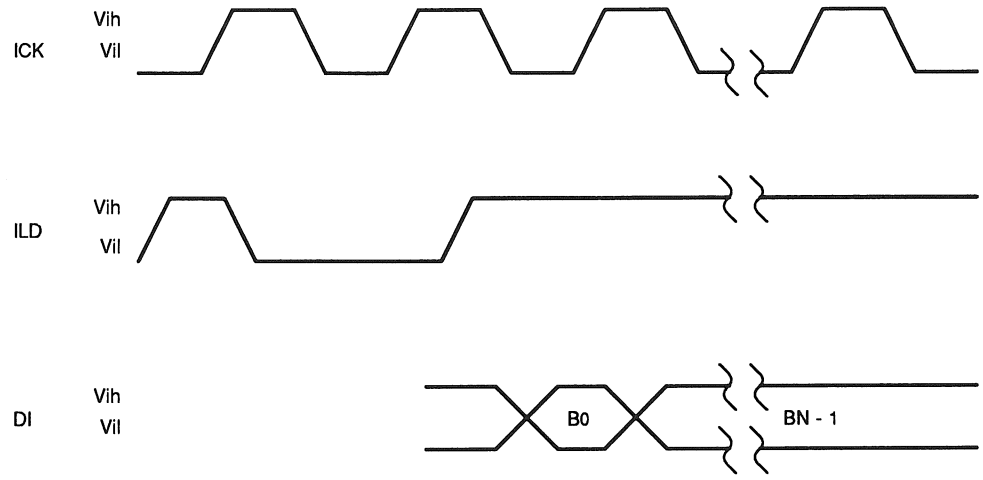



Figure 6. Serial Input Timing

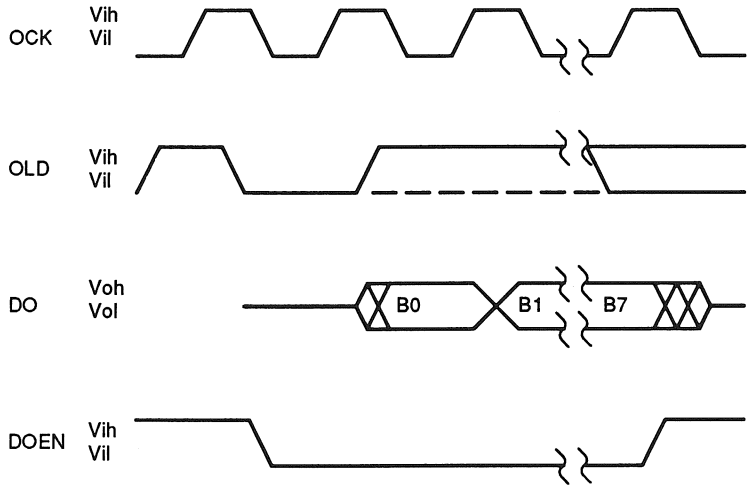



Figure 7. Serial Output Timing – 8 Bits

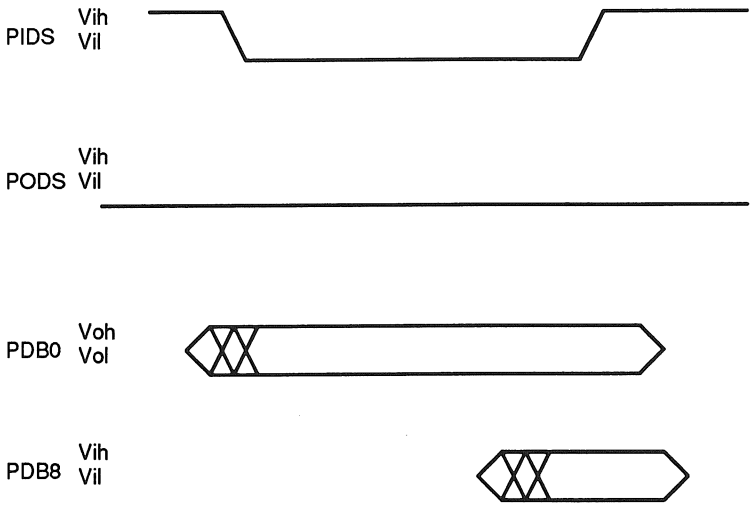



Figure 8. Parallel I/O Timing in S/C Mode

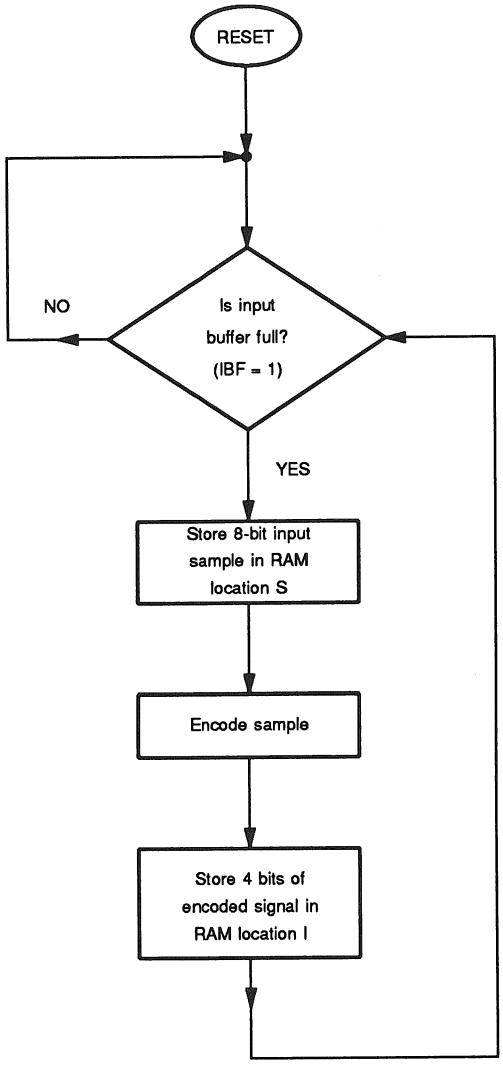



**Figure 9. A Flowchart for Implementing an Encoder in the ADPCM Demonstration System**

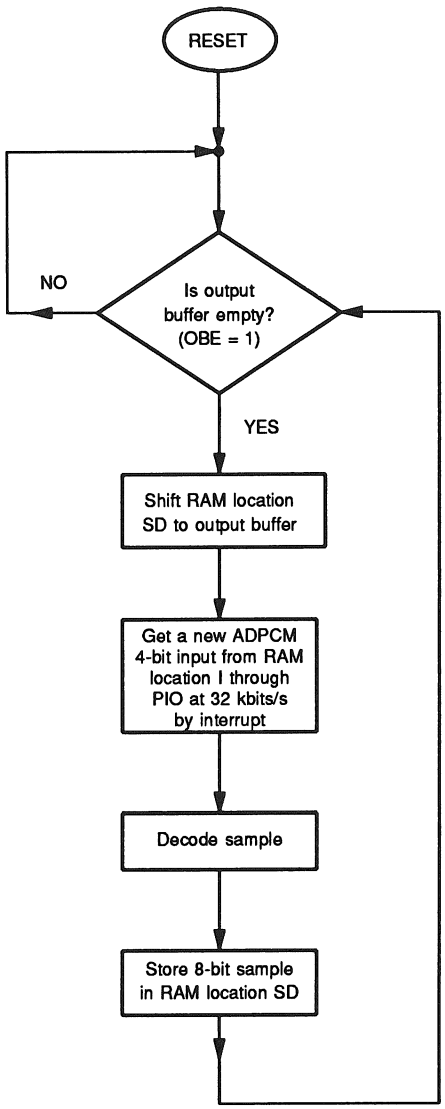



**Figure 10. A Flowchart for Implementing a Decoder in the ADPCM Demonstration System**

each DSP16 implements a full-duplex transcoder, allowing analog signals to be transmitted in both directions.

## Software Implementation

Figures 9 and 10 show the flowcharts for the ADPCM encoder and decoder, respectively. In this approach the encoder and decoder are synchronized with the codecs' 8 kHz sample clock. One bit is transferred through the parallel I/O port when the synchronous 32 kbits/s clock interrupts the DSP16 through PIDS. A flowchart of the interrupt service subroutine is shown in Figure 11.

Program listings 1 and 2 implement the CCITT ADPCM encoder and decoder, respectively.

Pages 12, 28, and 36 of this application note show the changes needed to implement the nonstandard algorithm. If nonstandard mode is selected, the native multiply instruction is used in several places, rather than the floating-point format described in the CCITT recommendation. This reduces the computational complexity by an amount that allows a full-duplex transcoder to be implemented on a single DSP16 running at a 75 ns instruction cycle time. The code headers correspond to the algorithm block diagram described earlier.

Table 4 shows the DSP16 RAM, ROM, and real-time utilization for both the CCITT and non-standard ADPCM algorithms.

**Table 4. DSP16 Utilization**

| Mode | | 16-bit words | | CPU Utilization |
|---|---|---|---|---|
| | | ROM (2K) | RAM (512) | at 55 ns |
| Standard | Encoder | 900 | 50 | 52% |
| | Decoder | 850 | 50 | 57% |
| Non-standard | Encoder/ Decoder | 1500 | 100 | 59% |

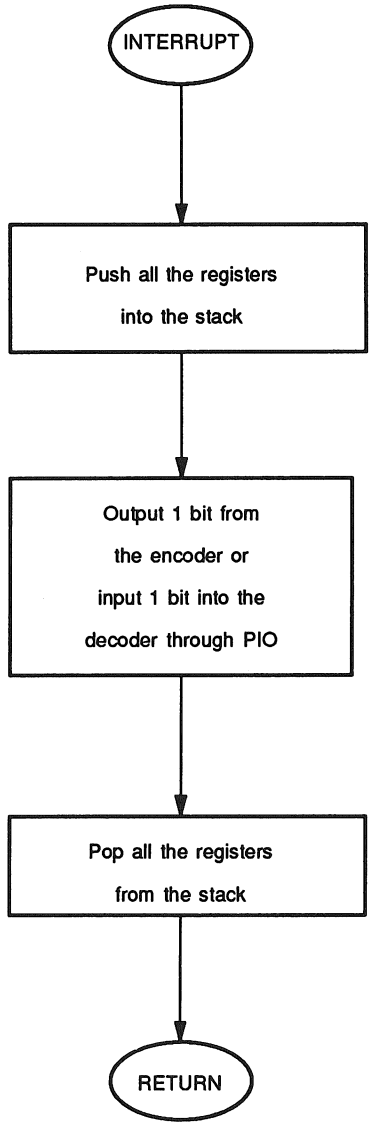



**Figure 11. A Flowchart of the Interrupt Service Subroutine**

### Program Listing 1. ADPCM Encoder Source Code

```
/*****************************************************************/
/*                                                               */
/* This program ADPCM encoder is coded according to the CCITT    */
/* standard algorithm described in G.721. All the notations used */
/* here are based on the CCITT documents and the program only    */
/* operates in u-law. The multiplication operations in the FMULT */
/* block are modified slightly to increase the DSP16 performance */
/* with the bit precision kept.  The code in the FMULT block is  */
/* fully compatible with the CCITT G.721 standard FMULT code.    */
/* The input data is assumed in RAM location SD before program   */
/* execution and the result is stored in RAM location I.         */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*      Program Initialization                                   */
/*                                                               */
/*****************************************************************/

adpcme: auc=0xc          /* Set auc register                     */
        y=0x0            /**         RESET SECTION              **/
        r1=YL            /* Set initial values for               */
        *r1++=y          /* Quantizer Scale Factor               */
        y=0x8800         /* Adaptation block.                    */
        *r1=y
        r2=YU
        y=544
        *r2=y
        y=0x0            /* Initialize the Adaptation            */
        r0=DML           /* Speed Control block.                 */
        *r0=y
        r1=DMS
        *r1=y
        r3=AP
        *r3=y
        r0=DQ1           /* Initialize the Adaptive              */
        r1=DQCNT1        /* and Reconstructed Signal             */
        y=1              /* Calculator block.                    */
        do 8 {
        *r0++=y
        *r1++=y
        }
        r0=temp1
        *r0=y
        y=0x0
        r0=PK1
        do 2 {
        *r0++=y
        }
        r0=B1
        r1=DQ1S
        do 8 {
        *r0++=y
        *r1++=y
        }
        r1=TD            /* Initialize the Tone and              */
        *r1=y            /* Transition Detector block.           */
```

```
/*********************************************************************/
/*                                                                   */
/*                 Main Program                                      */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*                 FMULT                                             */
/*                                                                   */
/*********************************************************************/

loop:
pred:   c0=-7               /** AFTER DELAY PARTS aparsc.s         **/
        r0=DQCNT1           /* Set the points before doing          */
        r1=DQ1              /* FMULT.                               */
        r2=WB1-1
        r3=B1
l0b18:  a0=*r3
        y=0xfffc
        a0=a0&y  *r2++=a1l
        if mi a0=-a0        /* a0 = |a0|                            */
        a0=a0>>1
        if eq a0h=a0h+1     /* If a0 = 0, then a0 = a0 + 1          */
        c1=-16              /* Set counter c1 = 16                  */
        a1=a0               /* a1 = a0                              */
        do 15 {             /* Compute exponent                     */
        ifc pl a1=a1<<1
        }
        a1=c2               /* Now c2 = -( exponent )               */
        a1=-a1              /* a1 = exponent part                   */
        j=a1
        y=tabl1
        a1=a1+y
        pt=a1
        y=a0 x=*pt++        /* y = |Bn| and x is a pattern          */
        a0=x                /* used to round |Bn|.                  */
        a0=a0&y  a1=*r3++
        a1=a1
        if mi a0=-a0        /* OK, a0 = 2's complement              */
        a1=*r0++            /* value of Bn with the same            */
        y=16                /* accuracy as standard.                */
        a1-y
        if pl a0=a0<<1
        x=a0
        a0=j
        y=a0
        a1=a1+y  y=*r1++    /* y = DQn                              */
        p=x*y               /* p = Bn * DQn                         */
        y=22
        a1-y                /* Compare a1 with 22 to set            */
        y=0x40              /* the right pattern with the           */
        move a0=y           /* table.                               */
        if pl a0=a0<<8      /* If a1 >= 22, set a0 = 0x400.         */
        a1=a1<<1            /* a1 = a1 * 2 for the offset           */
        y=tabl2
        a1=a1+y             /* a1 points to the table.              */
        pt=a1
        a1=p y=a0 x=*pt++
        p=x*y y=a0 x=*pt++
        a0=a1
        if mi a1=-a1
        a1=a1+p  p=x*y
        y=a1                /* Set y (32 bits) = a1.                */
        a1=a1<<16
```

```
        yl=a1
        a1=p
        a1=a1&y
        a1=a1>>8            /* Rescale a1.                          */
        a1=a1>>4
        a1=a1>>1
        a0=a0               /* Check the sign bit.                  */
        if mi a1=-a1        /* If negative, then a1 = -a1.          */
        if c0lt goto l0b18  /* Do eight times.                      */
        *r2++=a11

/*********************************************************************/
/*                                                                   */
/*                        ACCUM                                      */
/*                                                                   */
/*********************************************************************/

        r0=WB1              /**         ACCUM                      **/
        a0=*r0++
        y=*r0++
        do 5 {
        a0=a0+y y=*r0++     /* Sum for partial signal               */
        }                   /* estimate                             */
        a1=a0+y  y=*r0      /* a0 = SEZI                            */
        a1=a1+y             /* a1 = SEI                             */
        r0=SEZ              /* SEZ is 16 bits TC                    */
        a0=a0>>1
        *r0=a0
        r1=SE
        a1=a1>>1
        r3=AP               /** AFTER DELAY LIMA adspcl.s          **/

/*********************************************************************/
/*                                                                   */
/* For the non-standard CCITT ADPCM mode, the program from           */
/* the label, loop (Main program), to the previous line              */
/* (r3=AP) replaced by the following program segment.                */
/*                                                                   */
/* loop:                                                             */
/* pred:  r1=DQ1               ; aparsc                              */
/*        r2=B1                ; Set the points before               */
/*        y=0                  ; doing FMULT                         */
/*        a1=y  x=*r2++        ; Clear a1                            */
/*        do 6 {                                                     */
/*        y=*r1++                                                    */
/*        p=x*y                ; p = DQn * Bn                        */
/*        a0=p  x=*r2++        ; a0 = DQn * Bn                       */
/*        a0=a0>>1                                                   */
/*        a0=a0<<4             ; Scaling                             */
/*        y=a0                                                       */
/*        a1=a1+y                                                    */
/*        }                                                          */
/*        a0=a1>>1             ; a0 = SEZ                            */
/*        r3=SEZ                                                     */
/*        *r3=a0               ; Store it                            */
/*        redo 2                                                     */
/*        a1=a1>>1             ; a1 = SE                             */
/*        r1=SE                                                      */
/*        r3=AP                ; adspcl.s                            */
/*                                                                   */
/*********************************************************************/

        a0=*r3
        do 2 {
        a0=a0>>1
        }
```

```
        y=0x3f              /* y = ( 255 >> 2 )                        */
        yl=0xc00
        a0-y  *r1=a1        /* SE is 16 bits TC                        */
        y=64
        if gt a0=y
        r2=AL
        *r2=a0
        r2=YU               /** AFTER DELAY  MIX  qsfadp.s            **/
        r1=YL
        a1=*r1++
        a1l=*r1--
        a1=a1<<8
        a1=a1<<1
        a1=a1<<1
        move y=a1           /* y = (YL >> 6)                           */
        a0=*r2
        a0=a0-y             /* a0 = DIFM                               */
        r0=AL
        x=*r0
        y=a0
        p=x*y               /* p = PRODM = (DIFM * AL) >> 6.           */
        a0=p
        a0=a0<<8
        a0=a0<<1
        a0=a0<<1
        move y=a0           /* y = PROD                                */
        a1=a1+y
        y=0x1fff
        a1=a1&y
        r2=Y                /* Y = Standard Y +- 1 if (YL>>6) > YU     */
        *r2=a1

/*************************************************************************/
/*                                                                       */
/*                        EXPAND                                         */
/*                                                                       */
/*************************************************************************/

ipcdsc: r1=S                /**         EXPAND  ipcmv.s               **/
        a0=*r1
        y=0xff              /* Invert bits                             */
        a0=a0^y             /* This program only considers             */
        a1=a0>>4            /* the input as u-law.                     */
        y=0x7
        a1=a1&y             /* a1 = L = abc                            */
        a1=-a1
        c0=a1
        y=0xf
        a1=a0&y             /* a1 = V = xyzw                           */
        a1=a1<<1
        y=0x21
        a1=a1|y
        do 7 {
        if c0lt a1=a1<<1
        }
        y=33
        a1=a1-y
        y=0x80              /* Check the sign bit.                     */
        r0=SE
        a0=a0&y y=*r0
        if ne a1=-a1        /* If negative, then a1 = -a1.             */
        r1=D                /**         SUBTA                         **/
        a1=a1-y
        *r1=a1
```

```
/*****************************************************************/
/*                                                               */
/*                          LOG                                  */
/*                                                               */
/*****************************************************************/

adaqu1: r0=D                  /**         LOG  adaqua.s          **/
        y=*r0
        a0=y  *r0
        if mi a0=-a0          /* Convert D from 2's complement    */
        c1=-15                /* to signed magnitude              */
        a1=a0
        do 15 {
        ifc pl a1=a1<<1       /* Compute exponent                 */
        }
        y=0x7f00
        a0=a1&y
        a0=a0>>8
        a1=c2
        y=a0
        a1=-a1
        a1=a1<<8
        a1=a1>>1              /* a1 = EXP << 7                    */
        r0=Y
        a1=a1+y  a0=*r0       /* a1 = (EXP << 7) + MANT           */
        a0=a0>>1
        a0=a0>>1              /* a0 = Y >> 2                      */
        y=a0
        r0=YSH2
        a1=a1-y  *r0=a0       /**         SUBTB                  **/
        y=4095
        a1=a1&y
        r0=DLN                /* DLN = (DL + 4096 - y) & 4095     */
        *r0=a1
quan:   r0=DLN                /**         QUAN  adaqua.s         **/
        a1=*r0
        a1=a1>>16
        r3=temp1
        r1=0
        pt=dln
        y=*r3  x=*pt++        /* Compare a1 with DLN to find      */
l6b1:   p=x*y  y=*r3  x=*pt++    /*  the point to I.               */
        a0=a1-p  *r1++
        if gt goto l6b1
        r0=D
        y=*r0
        a0=y  *r0             /* Check the sign bit.              */
        y=neg-1
        if mi goto l6b3
        y=pos-1
l6b3:   a0=r1
        a0=a0+y
        pt=a0                 /* a0 points to I.                  */
        y=*r0  x=*pt++
        r0=I
        *r0=x
```

```
/*****************************************************************/
/*                                                               */
/*                          RECONST                              */
/*                                                               */
/*****************************************************************/

inadq1: r1=YSH2             /**     RECONST  inadqu.s          **/
        r2=I
        a0=*r2
        y=0xf
        a0=a0^y
        y=0x08
        a0&y y=*r2
        if ne a0=y
        y=dqln
        a0=a0+y             /* a0 is the point to DQLN table     */
        pt=a0
        y=*r1  x=*pt++
        a0=x                /**         ADDA                  **/
        a0=a0+y             /* DQL = (DQLN + (Y>>2)) & 4095      */
        y=0x7f              /**         ANTILOG               **/
        a1=a0&y             /* a1 = DMN                          */
        y=0x80              /* y = (1<<7)                        */
        a1=a1+y             /* a1 = DQT                          */
        a1=a1>>1
        a1=a1<<8
        y=0
        a0=a0<<4
        if mi a1=y          /* If DS=1, then DQMAG = 0.          */
        a0=a0>>16
        a0=a0<<4
        a0=a0<<1
        y=0xf
        a0=a0&y
        y=-14
        a0=a0+y
        c1=a0
        do 14 {
        if c1lt a1=a1>>1
        }
        r0=DQ0S
        a0=*r2              /* a0 = I                            */
        y=0x08
        a0=a0&y
        a0=a0<<1            /* Check DQS. If positive, then      */
        a0=a0>>4            /* set DQ0S=0, otherwise DQ0S=1.     */
        *r0=a0
        if ne a1=-a1        /* a1 = | a1 |                       */
        r0=DQ16
        *r0=a1
trans1: r0=YL               /** AFTER DELAY TRANS totrde.s     **/
        a0=*r0++
        a0l=*r0--           /* a0 = YL                           */
        a0=a0<<1
        x=a0
        a1=x                /* a1 = YLINT                        */
        a1=-a1
        c0=a1
        a1=a0<<4
        a1=a1<<1
        y=31
        a1=a1&y             /* a1=YLFRAC=(YL >> 10) & 31         */
        y=32
        a1=a1+y             /* a1=THR1=(32 + YLFRAC)             */
        do 8 {
```

```
        if c0lt a1=a1<<1
        }
        a0=x
        y=8
        a0-y                /* YLINT > 8 ?                              */
        y=0x3e00            /* y = 31 << 9                              */
        if gt a1=y          /* Yes, YLINT > 8.                          */
        a0=a1>>1
        y=a0
        a1=a1+y             /* a1 = THR2 + (THR2 >> 1)                  */
        a1=a1>>1            /* a1 = DQTHR                               */
        y=a1
        r1=DQ16
        a1=*r1
        a1=a1
        if mi a1=-a1
        r2=TD
        a1=a1-y  a0=*r2     /* a1 = DQMAG - DQTHR                       */
        if le a0=a0<<16     /* If less or equal, set                    */
        r3=TR               /* a0=0.                                    */
        *r3=a0

/*************************************************************************/
/*                                                                       */
/*                        ADDB                                           */
/*                                                                       */
/*************************************************************************/

apars1: r0=DQ16             /**  BEFORE DELAY -- ADDB  aparsc.s     **/
        a0=*r0
        r0=SE
        r3=SR1S
        y=*r0
        a0=a0+y  x=*r3--    /* a0 = SR                                  */
        r0=SR
        *r0=a0
        y=0                 /* Set a1 = SR sign bit                     */
        move a1=y
        if mi a1h=a1h+1     /**         FLOATB                       **/
        a0=a0
        if mi a0=-a0        /* a0 = MAG = | SR |                        */
        *r3++=x
        *r3=a1              /* Update the sign bit storage.             */
        a0=a0<<1
        if eq a0h=a0h+1
        c1=-16
        c2=-16
        a1=a0
        do 15 {             /* Find the EXP from the MAG.               */
        ifc pl a1=a1<<1
        }
        r2=SRCNT1
        a1=c2
        a1=-a1
        y=16
        a1-y  x=*r2         /* Check EXP = 16 ?                         */
        *r2--=a1
        if eq a0=a0>>1      /* Yes, shift it back.                      */
        y=tabl1             /* Use the table looking up to              */
        *r2=x               /* find the pattern for retaining           */
        a1=a1+y             /* the bit precision in                     */
        pt=a1               /* converting into floating                 */
        y=*r2 x=*pt++       /* format, and be represented               */
        a1=x                /* in signed magnitude.                     */
        y=a1
        a0=a0&y  a1=*r3
```

```
a1=a1
if ne a0=-a0
r2=SR1              /* Update the SR's parameters          */
y=*r2--
*r2++=y
*r2=a0
r0=DQ1              /**          UPB                      **/
r3=B1               /* Set the points for cache.          */
r1=DQ16
i=128
r2=TR
do 6 {              /* Do six times.                      */
a0=*r1              /* a0 = DQ                            */
a1=a0
y=*r0++             /* y = DQn                            */
a1=a1^y  y=*r3      /* a1 = Un = DQ ^ DQn                 */
a1=i                /* If a1 positive, then Un=0.         */
if mi a1=-a1        /* If a1 negative, then Un=1.         */
a0=a0
if eq a1=a1<<16     /* a1 = UGBn                          */
a1=a1+y  a0=*r3     /* a1 = UGBn + Bn                     */
a0=a0>>8            /* a0 = Bn >> 8                       */
y=a0
a1=a1-y  a0=*r2     /* a1 = BnP                           */
a0=a0               /**          TRIGB                   **/
if ne a1=a1<<16     /* If =| 0, set a1=0.                 */
*r3++=a1
}
r3=DQ16             /**          FLOATA                  **/
a0=*r3              /* a0 = DQ                            */
a0=a0<<1
if mi a0=-a0        /* a0 = |a0|                          */
if eq a0h=a0h+1
c1=-16              /* Set the initial counter to         */
a1=a0               /* find the EXP of a0.                */
do 15 {
ifc pl a1=a1<<1
}
a1=c2
a1=-a1              /* a1 = EXP of DQ                     */
r2=DQCNT0
y=tabl1
a1=a1+y *r2=a1
pt=a1
y=*r2 x=*pt++
a1=x
y=a1
a0=a0&y             /* a0 contains the magnitude DQ       */
r0=DQ0S             /* Check the sign bit of DQ0.         */
a1=*r0
a1=a1
if ne a0=-a0        /* Convert it into 2's complement.    */
r2=DQ0
*r2=a0
y=*r3               /**          ADDC                    **/
r0=SEZ
a0=*r0
a0=a0+y             /* a0 = DQSEZ                         */
y=0
move a1=y           /* If DQSEZ=0, set a1 = 1             */
if eq a1h=a1h+1     /* otherwise, set a1 = 0              */
r0=SIGPK
*r0=a1              /* Store it in SIGPK.                 */
move a1=y
a0=a0
```

```
        if mi a1h=a1h+1     /* Find the PK0                          */
        r2=PK0
        *r2=a1
        r0=A1               /**         UPA2                      **/
        y=*r0
        a0=y *r0            /* Check A1S = 0 ?                       */
        if mi goto lab26
        y=0x1fff            /* Yes, compare with 8191.               */
        a0-y
        if gt a0=y          /* If a0 > 8191, set a0 = 8191           */
        a0=a0>>16
        a0=a0<<1
        a0=a0<<1            /* a0 = a0 >> 14                         */
        goto lab28
lab26:  y=0x0               /* No, compare with 57345                */
        yl=*r0
        a0=y
        yl=0xe000
        a1=a0-y             /* A1 - 57344 >= 0 ?                     */
        a0=a0<<1
        a0=a0<<1
        y=0x1
        yl=0xffff
        a0=a0&y             /* Yes, a0 = A1 << 2                     */
        yl=0x8004
        a1=a1
        if le a0=y          /* No, a0 = 24577 << 2                   */
lab28:  y=*r2
        r1=PK1
        a1=y y=*r1 x=*pt++
        a1=a1^y             /* a1 = PK0 ^ PK1 = PKS1                 */
        if eq a0=-a0        /* a0 = FA                               */
        y=*r2
        r1=PK2
        a1=y y=*r1 x=*pt++
        a1=a1^y             /* a1 = PK0 ^ PK2 = PKS2                 */
        y=0x1               /* Set y = 114688 if a1 < 0              */
        yl=0xc000
        if ne goto lab31
        y=0x0               /* otherwise, set y = 16384              */
        yl=0x4000
lab31:  a0=a0+y             /* a0 = UGA2B                            */
        r0=SIGPK
        a0=a0>>1
        a0=a0<<16
        a0=a0>>4
        a0=a0>>1
        a0=a0>>1            /* a0 = a0 << 9                          */
        y=0x0
        a1=*r0
        a1=a1               /* Check SIGPK = 0 ?                     */
        if ne a0=y          /* If not, set a0 = 0.                   */
        r0=A2               /* a0 = UGA2                             */
        y=*r0
        a0=a0+y             /* a0 = UGA2 + A2                        */
        a1=y                /* a1 = A2                               */
        a1=a1>>8
        a1=a1<<1            /* a1 = A2 >> 7                          */
        y=a1
        a0=a0-y             /* a0 = A2T                              */
        a1=a0               /**         LIMC                      **/
        if mi a1=-a1        /* a1 = |a1|                             */
        y=0x3000            /* Set y = 12288                         */
        a1-y                /* |a1| > 12288 ?                        */
        if pl a1=y          /* Yes, set |a1| = 12288                 */
```

```
a0=a0
if mi a1=-a1          /* a1 = A2P                              */
r0=A2P
*r0=a1
r1=TR                 /**          TRIGB                      **/
a0=*r1
a0=a0
if ne a1=a1<<16       /* If TR =| 0, set a1 = 0.              */
r1=A2
*r1=a1
r1=PK1                /**          UPA1                       **/
r0=PK0
a1=*r1++
y=*r0
a1=a1^y  *r1--=a1     /* a1 = PK0 ^ PK1 = PKS                 */
*r1=y
y=192
move a0=y             /* Set a0 = 192 if PKS = 0              */
if ne a0=-a0          /* otherwise, set a0 = 65344            */
r0=SIGPK
a1=*r0
a1=a1
if ne a0=a0<<16       /* a0=UGA1                              */
r0=A1
y=*r0                 /* y = A1                               */
a0=a0+y               /* a0 = UGA1 + A1                       */
a1=y                  /* a1 = A1                              */
a1=a1>>8              /* a1 = A1 >> 8                         */
y=a1
a0=a0-y               /* a0 = UGA1 + A1 - (A1 >> 8)           */
r2=A1T
*r2=a0                /* a0 = A1T                             */
y=0x3c00              /**          LIMD                       **/
r1=A2P                /* Set y = 15360                        */
a1=*r1
a1=a1-y               /* a1 = A2P - 15360                     */
a1=-a1
y=a1                  /* y = A1UL                             */
a1=a0
if mi a1=-a1          /* a1 = | A1T |                         */
a1-y                  /* | A1T | - A1UL                       */
if pl a1=y
a0=a0
if mi a1=-a1          /* a1 = A1P                             */
r0=TR                 /**          TRIGB                      **/
a0=*r0
a0=a0                 /* Check TR = 0 ?                       */
if ne a1=a1<<16       /* If not, then set a1 = 0              */
r0=A1
*r0=a1                /* otherwise, keep a1.                  */
r0=DQ0
y=*r0++
do 6 {
*r0zp:y               /* Update the DQn parameters            */
}                     /* for the next input data.             */
r0=DQ0S
y=*r0++
do 6 {
*r0zp:y               /* Update the sign bit of DQn           */
}                     /* for the next input data.             */
r0=DQCNT0
y=*r0++
do 6 {
*r0zp:y               /* Update the parameters for            */
}                     /* the next input data.                 */
```

```
/*****************************************************************/
/*                                                               */
/*                          TONE                                 */
/*                                                               */
/*****************************************************************/

totrd1: r0=A2P              /**         TONE  totrde.s          **/
        a0=*r0
        y=0x2e00            /* y = 11776                         */
        a0=a0+y             /* a2(k) + 0.71875                   */
        y=0
        move a1=y
        if mi a1h=a1h+1     /* a1 = TDP                          */
        r0=TDP
        *r0=a1
        r3=TR               /**         TRIGB                   **/
        a0=*r3
        a0=a0               /* Check TR = 0 ?                    */
        *r1=a1
adspc1: r0=I                /**         FUNCTF  adspcl.s        **/
        a0=*r0              /* a0 = I                            */
        y=FI
        a0=a0+y             /* Use the table looking up to       */
        r2=DML              /* find the FI (scaled) value.       */
        pt=a0
        y=*r2 x=*pt++
        a0=x                /* a0 = FI                           */
        a1=a0<<1            /**         FILTB                   **/
        a1=a1<<1            /* a1 = FI << 11                     */
        a1=a1-y             /* a1 = (FI<<11) - DML = DIF         */
        a1=a1<<1
        a1=a1>>8            /* a1 = DIF >> 7 = DIFSX             */
        a1=a1+y             /* a1 = DIFSX + DML                  */
        y=0x3fff            /* Set y = 16383                     */
        r1=DMS
        a1=a1&y y=*r1
        a0=a0-y *r2=a1      /**         FILTA                   **/
        a0=a0>>1
        a0=a0<<4
        a0=a0>>8            /* a0 = DIFSX = DIF >> 5             */
        a0=a0+y             /* a0 = DIFSX + DMS                  */
        y=0xfff             /* Set y = 4095                      */
        a0=a0&y y=*r2
        *r1=a0              /* a0 = DMSP                         */
        a0=a0<<1            /**         SUBTC                   **/
        a0=a0<<1            /* a0 = DMSP << 2                    */
        a0=a0-y  a1=*r2     /* a0 = (DMSP<<2) - DMLP = DIF       */
        if mi a0=-a0        /* a0 = | DIF | = DIFM               */
        a1=a1>>4
        a1=a1<<1            /* a1 = DTHR = DMLP >> 3             */
        move y=a1
        r3=Y
        a0=a0-y  a1=*r3     /* a0 = DIFM - DTHR                  */
        r3=TDP
        a0=*r3
        y=1
        if pl a0=y
        y=1536              /* Set y = 1536                      */
        a1=a1-y             /* a1 = Y - 1536                     */
        y=1
        if mi a0=y          /* a0 = AX                           */
        a0=a0<<8            /**         FILTC                   **/
        a0=a0<<1            /* a0 = AX << 9                      */
        r3=AP
        y=*r3               /* y = AP                            */
```

```
        a0=a0-y             /* a0 = (AX<<9) - AP = DIF              */
        a0=a0>>4            /* a0 = DIFSX                           */
        a0=a0+y             /* a0 = DIFSX + AP                      */
        y=0x3ff
        r2=TR               /**          TRIGA                     **/
        a0=a0&y y=*r2       /* a0 = APP                             */
        a1=y *r2
        y=0x100             /* Set y = 256                          */
        if ne a0=y
        *r3=a0              /* a0 = APR                             */

/*********************************************************************/
/*                                                                   */
/*                        FUNCTW                                     */
/*                                                                   */
/*********************************************************************/

qsfad1: r0=I                /**          FUNCTW  qsfadp.s          **/
        a0=*r0
        y=WI
        a0=a0+y             /* a0 points to WI                      */
        pt=a0
        y=*r0 x=*pt++       /* x = WI                               */
        a0=x                /**          FILTD                     **/
        a0=a0>>8
        a0=a0>>4
        a0=a0<<1            /* a0l = WI << 5                        */
        r2=Y
        y=0x0
        yl=*r2              /* y = Y                                */
        a0=a0-y             /* a0 = DIF                             */
        a0=a0>>4
        a0=a0>>1            /* a0 = DIFSX = DIF >> 5                */
        a0=a0+y             /* a0 = Y + DIFSX                       */
        a0=a0<<16
        y=0x1fff            /* Set y = 8191                         */
        a0=a0&y             /* a0 = YUT                             */
        y=544               /**          LIMB                      **/
        a0-y                /* If negative, GELL = 1                */
        if mi a0=y
        y=5120              /* Set y = 5120                         */
        a0-y                /* If positive, GEUL = 0                */
        if pl a0=y          /* a0 = YUP                             */
        r3=YU
        *r3=a0              /* Store it.                            */
        y=0
        yl=a0
        r1=YL               /**          FILTE                     **/
        a0=*r1++
        a0l=*r1--           /* a0 = YL                              */
        a0=-a0              /* a0 = -YL                             */
        a0=a0>>4
        a0=a0>>1
        a0=a0>>1            /* a0 = ( -YL ) >> 6                    */
        a0=a0+y             /* a0 = DIF = DIFSX                     */
        y=*r1++
        yl=*r1--            /* y = YL                               */
        a0=a0+y             /* a0 = YL + DIFSX                      */
        y=0x7
        yl=0xffff
        a0=a0&y             /* a0 = YLP                             */
        *r1++=a0
        *r1=a0l             /* Store it.                            */
        goto loop           /* Go back to execute the next          */
                            /* sampling data.                       */
```

```
/*****************************************************************/
/*                                                               */
/*                End of Program                                 */
/*                                                               */
/*****************************************************************/

posx:   int         0x9
        int         0xa
        int         0xb
        int         0xc
        int         0xd
        int         0xe
        int         0xf
        int         0x7
        int         0x9
negx:   int         0x6
        int         0x5
        int         0x4
        int         0x3
        int         0x2
        int         0x1
        int         0x0
        int         0x7
        int         0x6
WI:     int         0xfff4      /* -12 = 4084 for 12 bits          */
        int         18
        int         41
        int         64
        int         112
        int         198
        int         355
        int         1122
        int         1122
        int         355
        int         198
        int         112
        int         64
        int         41
        int         18
        int         0xfff4
FI:     int         0
        int         0
        int         0
        int         0x200
        int         0x200
        int         0x200
        int         0x600
        int         0xe00
        int         0xe00
        int         0x600
        int         0x200
        int         0x200
        int         0x200
        int         0
        int         0
        int         0
pos:    int         0x01
        int         0x02
        int         0x03
        int         0x04
        int         0x05
        int         0x06
        int         0x07
        int         0x0f
        int         0x01
```

```
neg:    int         0x0e
        int         0x0d
        int         0x0c
        int         0x0b
        int         0x0a
        int         0x09
        int         0x08
        int         0x0f
        int         0x0e
dln:    int         79
        int         177
        int         245
        int         299
        int         348
        int         399
        int         2047
        int         3971
        int         4095
dqln:   int         2048
        int         4
        int         135
        int         213
        int         273
        int         323
        int         373
        int         425
tabl1:  int         0x0
        int         0x3f
        int         0x3f
        int         0x3f
        int         0x3f
        int         0x3f
        int         0x3f
        int         0x7e
        int         0xfc
        int         0x1f8
        int         0x3f0
        int         0x7e0
        int         0xfc0
        int         0x1f80
        int         0x3f00
        int         0x7e00
        int         0x7e00
tabl2:  28*int         0x0
        int         0x3
        int         0x7fff
        int         0x6
        int         0x7fff
        int         0xc
        int         0x7fff
        int         0x18
        int         0x7fff
        int         0x30
        int         0x7fff
        int         0x60
        int         0x7fff
        int         0xc0
        int         0x7fff
        int         0x180
        int         0x7fff
        int         0x3
        int         0x6fff
        int         0x6
        int         0x6ffe
        int         0xc
```

```
        int         0x6ffc
        int         0x18
        int         0x6ff8
        int         0x30
        int         0x6ff0
        int         0x60
        int         0x6fe0
        int         0xc0
        int         0x6fc0
        int         0x180
        int         0x6f80
        int         0x300
        int         0x6f00

.ram
SD:     int
DLN:    int
S:      int
SL:     int
D:      int
temp:   int
temp1:  int
Y:      int
YSH2:   int
I:      int
R:      int
YL:     int
        int
YU:     int
AL:     int
TDP:    int
AP:     int
DML:    int
DMS:    int
DQ0:    int
DQ1:    int
DQ2:    int
DQ3:    int
DQ4:    int
DQ5:    int
DQ6:    int
SR2:    int
SR1:    int
B1:     int
B2:     int
B3:     int
B4:     int
B5:     int
B6:     int
A2:     int
A1:     int
        int
WB1:    int
WB2:    int
WB3:    int
WB4:    int
WB5:    int
WB6:    int
WA2:    int
WA1:    int
PK1:    int
PK2:    int
PK0:    int
A2P:    int
SE:     int
```

```
SEZ:     int
DQ16:    int             /* The TC of DQ                                  */
TR:      int
A1T:     int
SIGPK:   int
SR:      int
TD:      int
DQ0S:    int
DQ1S:    int
DQ2S:    int
DQ3S:    int
DQ4S:    int
DQ5S:    int
DQ6S:    int
SR2S:    int
SR1S:    int
DQCNT0:  int
DQCNT1:  int
DQCNT2:  int
DQCNT3:  int
DQCNT4:  int
DQCNT5:  int
DQCNT6:  int
SRCNT2:  int
SRCNT1:  int
.endram
```

### Program Listing 2. ADPCM Decoder Source Code

```
/*****************************************************************/
/*                                                               */
/* This program ADPCM decoder is coded according to the          */
/* CCITT standard algorithm described in G.721.                  */
/* All the notations used here are based on the CCITT            */
/* documents and the program only operates in u-law.             */
/* the multiplication operations in the FMULT block are          */
/* modified slightly to increase the DSP16 performance           */
/* with the bit precision kept. The code in the FMULT            */
/* block is fully compatible with the CCITT G.721                */
/* standard FMULT code. The input data is assumed in             */
/* RAM location I before program execution and the               */
/* result is stored in RAM location SD.                          */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*        Program Initialization                                 */
/*                                                               */
/*****************************************************************/

adpcmd: auc=0xc         /* Set auc register                      */
        y=0x0           /**        RESET SECTION                 */
        r1=YL           /* Set initial values for               */
        *r1++=y         /* Quantizer Scale Factor               */
        y=0x8800        /* Adaptation block.                    */
        *r1=y
        r2=YU
        y=544
        *r2=y
        y=0x0           /* Initialize the Adaptation            */
        r0=DML          /* Speed Control block                  */
        *r0=y
        r1=DMS
        *r1=y
        r3=AP
        *r3=y
        r0=DQ1          /* Initialize the Adaptive              */
        r1=DQCNT1       /* and Reconstructed Signal             */
        y=1             /* Calculator block                     */
        do 8 {
        *r0++=y
        *r1++=y
        }
        r0=temp1
        *r0=y
        y=0x0
        r0=PK1
        do 2 {
        *r0++=y
        }
        r0=B1
        r1=DQ1S
        do 8 {
        *r0++=y
        *r1++=y
        }
        r1=TD            /* Initialize the Tone and             */
        *r1=y            /* Transition Detector block.          */
```

```
/*********************************************************************/
/*                                                                   */
/*                 Main Program                                      */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*                 FMULT                                             */
/*                                                                   */
/*********************************************************************/

loop:
pred:   c0=-7           /** AFTER DELAY PARTS aparsc.s              **/
        r0=DQCNT1       /* Set the points before doing              */
        r1=DQ1          /* FMULT.                                   */
        r2=WB1-1
        r3=B1
l0b18:  a0=*r3
        y=0xfffc
        a0=a0&y  *r2++=a1l
        if mi a0=-a0        /* a0 = |a0|                            */
        a0=a0>>1
        if eq a0h=a0h+1     /* If a0=0, then a0 = a0 + 1            */
        c1=-16              /* Set counter c1 = 16                  */
        a1=a0               /* a1 = a0                              */
        do 15 {             /* Compute exponent                     */
        ifc pl a1=a1<<1
        }
        a1=c2               /* Now c2 = -( exponent )               */
        a1=-a1              /* a1 = exponent part                   */
        j=a1
        y=tabl1
        a1=a1+y
        pt=a1
        y=a0 x=*pt++        /* y = |Bn| and x is a pattern          */
        a0=x                /* used to round |Bn|.                  */
        a0=a0&y  a1=*r3++
        a1=a1
        if mi a0=-a0        /* OK, a0 = two's complement            */
        a1=*r0++            /* value of Bn with the same            */
        y=16                /* accuracy as standard.                */
        a1-y
        if pl a0=a0<<1
        x=a0
        a0=j
        y=a0
        a1=a1+y y=*r1++     /* y = DQn                              */
        p=x*y               /* p = Bn * DQn                         */
        y=22
        a1-y                /* Compare a1 with 22 to set            */
        y=0x40              /* the right pattern with the           */
        move a0=y           /* table.                               */
        if pl a0=a0<<8      /* If a1 >= 22, set a0 = 0x400.         */
        a1=a1<<1            /* a1 = a1 * 2 for the offset           */
        y=tabl2
        a1=a1+y             /* a1 points to the table.              */
        pt=a1
        a1=p y=a0 x=*pt++
        p=x*y y=a0 x=*pt++
        a0=a1
        if mi a1=-a1
        a1=a1+p  p=x*y
        y=a1                /* Set y (32 bits) = a1.                */
        a1=a1<<16
```

```
        yl=a1
        a1=p
        a1=a1&y
        a1=a1>>8            /* Rescale a1.                          */
        a1=a1>>4
        a1=a1>>1
        a0=a0               /* Check the sign bit.                  */
        if mi a1=-a1        /* If negative, then a1 = -a1.          */
        if c0lt goto l0b18  /* Do eight times.                      */
        *r2++=a1l

/*********************************************************************/
/*                                                                   */
/*                         ACCUM                                     */
/*                                                                   */
/*********************************************************************/

        r0=WB1              /**         ACCUM                      **/
        a0=*r0++
        y=*r0++
        do 5 {
        a0=a0+y y=*r0++     /* Sum for partial signal               */
        }                   /* estimate                             */
        a1=a0+y  y=*r0      /* a0 = SEZI                            */
        r0=SEZ              /* SEZ is 16 bits TC                    */
        a0=a0>>1
        a1=a1+y  *r0=a0     /* a1 = SEI                             */
        r1=SE
        a1=a1>>1
        r3=AP               /** AfTER DELAY LIMA adspcl.s         **/

/*********************************************************************/
/*                                                                   */
/* For the non-standard CCITT ADPCM mode, the program from           */
/* the label, loop (Main program), to the previous line              */
/* (r3=AP) replaced by the following program segment.                */
/*                                                                   */
/* loop:                                                             */
/* pred:  r1=DQ1               ; aparsc                              */
/*        r2=B1                ; Set the points before               */
/*        y=0                  ; doing FMULT                         */
/*        a1=y  x=*r2++        ; Clear a1                            */
/*        do 6 {                                                     */
/*        y=*r1++                                                    */
/*        p=x*y                ; p = DQn * Bn                        */
/*        a0=p  x=*r2++        ; a0 = DQn * Bn                       */
/*        a0=a0>>1                                                   */
/*        a0=a0<<4             ; Scaling                             */
/*        y=a0                                                       */
/*        a1=a1+y                                                    */
/*        }                                                          */
/*        a0=a1>>1             ; a0 = SEZ                            */
/*        r3=SEZ                                                     */
/*        *r3=a0               ; Store it                            */
/*        redo 2                                                     */
/*        a1=a1>>1             ; a1 = SE                             */
/*        r1=SE                                                      */
/*        r3=AP                ; adspcl.s                            */
/*                                                                   */
/*********************************************************************/

        a0=*r3
        a0=a0>>1
        a0=a0>>1
        y=0x3f              /* y = ( 255 >> 2 )                     */
        yl=0xc00
```

```
        a0-y  *r1=a1        /* SE is 16 bits TC                    */
        y=64
        if gt a0=y
        x=a0                /* AL = x = a0                         */
        r2=YU               /** AFTER DELAY MIX qsfadp.s           **/
        r0=YL
        a1=*r0++
        a1l=*r0--
        a1=a1<<8
        a1=a1<<1
        a1=a1<<1
        move y=a1           /* y = (YL >> 6)                       */
        a0=*r2              /* a0 = YU                             */
        a0=a0-y             /* a0 = DIFM                           */
        y=a0
        p=x*y               /* p = PRODM = (DIFM * AL) >> 6.       */
        a0=p
        a0=a0<<8
        a0=a0<<1
        a0=a0<<1
        move y=a0           /* y = PROD                            */
        a1=a1+y
        y=0x1fff
        r2=I
        a1=a1&y  a0=*r2     /* a0 = I                              */
        r1=Y                /* Y = Standard Y +- 1 if (YL>>6) > YU */

/*********************************************************************/
/*                                                                   */
/*                        RECONST                                    */
/*                                                                   */
/*********************************************************************/

inadq1: r3=YSH2             /**       RECONST  inadqu.s           **/
        y=0xf
        a0=a0^y  *r1=a1
        y=0x08
        a0&y  y=*r2
        if ne a0=y
        y=dqln
        a0=a0+y  a1=*r1     /* a0 is the point to DQLN table       */
        pt=a0
        y=*r1  x=*pt++
        a0=x                /**         ADDA                      **/
        a1=a1>>1
        a1=a1>>1
        y=a1                /* y = a1 = Y >> 2                     */
        a0=a0+y  *r3=a1
        y=0x7f              /**         ANTILOG                   */
        a1=a0&y             /* a1 = DMN                            */
        y=0x80              /* y = (1 << 7)                        */
        a1=a1+y             /* a1 = DQT                            */
        a1=a1>>1
        a1=a1<<8
        a0=a0<<4
        if mi a1=a1<<16     /* If DS=1, then DQMAG = 0.            */
        a0=a0>>16
        a0=a0<<4
        a0=a0<<1
        y=0xf
        a0=a0&y
        y=-14
        a0=a0+y
        c1=a0
        do 14 {
        if c1lt a1=a1>>1
```

```
        }
        r1=DQ0S
        a0=*r2              /* a0 = I                                  */
        y=0x08
        a0=a0&y
        a0=a0<<1            /* Check DQS. If positive, then            */
        a0=a0>>4            /* set DQ0S=0, otherwise DQ0S=1.           */
        *r1=a0
        if ne a1=-a1        /* a1 = | a1 |                             */
        r1=DQ16
        *r1=a1
trans1: a0=*r0++            /** AFTER DELAY TRANS totrde.s            **/
        a0l=*r0--           /* a0 = YL                                 */
        a0=a0<<1
        x=a0
        a1=x                /* a1 = YLINT                              */
        a1=-a1
        c0=a1
        a1=a0<<4
        a1=a1<<1
        y=31
        a1=a1&y             /* a1=YLFRAC=(YL >> 10) & 31               */
        y=32
        a1=a1+y             /* a1=THR1=(32 + YLFRAC)                   */
        do 8 {
        if c0lt a1=a1<<1
        }
        a0=x
        y=8
        a0-y                /* YLINT > 8 ?                             */
        y=0x3e00            /* y = 31 << 9                             */
        if gt a1=y          /* Yes, YLINT > 8.                         */
        a0=a1>>1
        y=a0
        a1=a1+y  a0=*r1     /* a1=THR2+(THR2>>1), r1=DQ16              */
        a1=a1>>1            /* a1 = DQTHR                              */
        y=a1
        a0=a0
        if mi a0=-a0
        r2=TD
        a0=a0-y  a1=*r2     /* a0 = DQMAG - DQTHR                      */
        if le a1=a1<<16     /* If less or equal, set                   */
        r3=TR               /* a0=0.                                   */
        *r3=a1

/*****************************************************************/
/*                                                               */
/*                        ADDB                                   */
/*                                                               */
/*****************************************************************/

apars1: a0=*r1          /**  BEFORE DELAY -- ADDB  aparsc.s        **/
        r0=SE           /* r1=DQ16                                 */
        r3=SR1S
        y=*r0
        a0=a0+y  x=*r3--    /* a0 = SR                             */
        r0=SR
        *r0=a0
        y=0                 /* Set a1 = SR sign bit                */
        move a1=y
        if mi a1h=a1h+1     /**         FLOATB                    **/
        a0=a0
        if mi a0=-a0        /* a0 = MAG = | SR |                   */
        *r3++=x
        *r3=a1              /* Update the sign bit storage.        */
        a0=a0<<1
```

```
if eq a0h=a0h+1
c1=-16
c2=-16
a1=a0
do 15 {              /* Find the EXP from the MAG.          */
ifc pl a1=a1<<1
}
r2=SRCNT1
a1=c2
a1=-a1
y=16
a1-y  x=*r2          /* Check EXP = 16 ?                    */
*r2--=a1
if eq a0=a0>>1       /* Yes, shift it back.                 */
y=tabl1              /* Use the table looking up to         */
*r2=x                /* find the pattern for retaining      */
a1=a1+y              /* the bit precision in                */
pt=a1                /* converting into floating            */
y=*r2 x=*pt++        /* format, and be represented          */
a1=x                 /* in signed magnitude.                */
y=a1
a0=a0&y  a1=*r3
a1=a1
if ne a0=-a0
r2=SR1               /* Update the SR's parameters.         */
y=*r2--
*r2++=y
*r2=a0
r0=DQ1               /**          UPB                     **/
r3=B1                /* Set the points for cache.           */
i=128
r2=TR
do 6 {               /* Do six times.                       */
a0=*r1               /* a0 = DQ                             */
a1=a0
y=*r0++              /* y = DQn                             */
a1=a1^y  y=*r3       /* a1 = Un = DQ ^ DQn                  */
a1=i                 /* If a1 positive, then Un=0.          */
if mi a1=-a1         /* If a1 negative, then Un=1.          */
a0=a0
if eq a1=a1<<16      /* a1 = UGBn                           */
a1=a1+y  a0=*r3      /* a1 = UGBn + Bn                      */
a0=a0>>8             /* a0 = Bn >> 8                        */
y=a0
a1=a1-y  a0=*r2      /* a1 = BnP                            */
a0=a0                /**          TRIGB                   **/
if ne a1=a1<<16      /* If =| 0, set a1=0.                  */
*r3++=a1
}
a0=*r1               /**          FLOATA                  **/
a0=a0<<1             /* a0 = DQ, r1=DQ16                    */
if mi a0=-a0         /* a0 = |a0|                           */
if eq a0h=a0h+1
c1=-16               /* Set the initial counter to          */
a1=a0                /* find the EXP of a0.                 */
do 15 {
ifc pl a1=a1<<1
}
a1=c2
a1=-a1               /* a1 = EXP of DQ                      */
r2=DQCNT0
y=tabl1
a1=a1+y *r2=a1
pt=a1
y=*r2 x=*pt++
```

```
        a1=x
        y=a1
        r0=DQ0S
        a0=a0&y  a1=*r0     /* a0 contains the magnitude DQ        */
        a1=a1               /* Check the sign bit of DQ0.          */
        if ne a0=-a0        /* Convert it into 2's complement.     */
        r2=DQ0
        y=*r1               /**         ADDC                    **/
        r0=SEZ
        a1=*r0
        a1=a1+y  *r2=a0     /* a0 = DQSEZ                          */
        y=0
        move a0=y           /* If DQSEZ=0, set a0 = 1              */
        if eq a0h=a0h+1     /* otherwise, set a0 = 0               */
        r0=SIGPK
        *r0=a0              /* Store it in SIGPK.                  */
        move a0=y
        a1=a1
        if mi a0h=a0h+1     /* Find the PK0                        */
        r2=PK0
        r0=A1               /**         UPA2                    **/
        y=*r0
        a1=y *r2=a0         /* Check A1S = 0 ?                     */
        if mi goto lab26
        y=0x1fff            /* Yes, compare with 8191.             */
        a0-y
        if gt a0=y          /* If a0 > 8191, set a0 = 8191         */
        a0=a0>>16
        a0=a0<<1
        a0=a0<<1            /* a0 = a0 >> 14                       */
        goto lab28
lab26:  y=0x0               /* No, compare with 57345              */
        yl=*r0
        a0=y
        yl=0xe000
        a1=a0-y             /* A1 - 57344 >=0 ?                    */
        a0=a0<<1
        a0=a0<<1
        y=0x1
        yl=0xffff
        a0=a0&y             /* Yes, a0 = A1 << 2                   */
        yl=0x8004
        a1=a1
        if le a0=y          /* No, a0 = 24577 << 2                 */
lab28:  y=*r2
        r1=PK1
        a1=y y=*r1 x=*pt++
        a1=a1^y             /* a1 = PK0 ^ PK1 = PKS1               */
        if eq a0=-a0        /* a0 = FA                             */
        y=*r2
        r1=PK2
        a1=y y=*r1 x=*pt++
        a1=a1^y             /* a1 = PK0 ^ PK2 = PKS2               */
        y=0x1               /* Set y = 114688 if a1 < 0            */
        yl=0xc000
        if ne goto lab31
        y=0x0               /* otherwise, set y = 16384            */
        yl=0x4000
lab31:  r0=SIGPK
        a0=a0+y  a1=*r0     /* a0 = UGA2B                          */
        a0=a0>>1
        a0=a0<<16
        a0=a0>>4
        a0=a0>>1
        a0=a0>>1            /* a0 = a0 << 9                        */
```

```
y=0x0
a1=a1                  /* Check SIGPK = 0 ?                    */
if ne a0=y             /* If not, set a0 = 0.                  */
r0=A2                  /* a0 = UGA2                            */
y=*r0
a0=a0+y                /* a0 = UGA2 + A2                       */
a1=y                   /* a1 = A2                              */
a1=a1>>8
a1=a1<<1               /* a1 = A2 >> 7                         */
y=a1
a0=a0-y                /* a0 = A2T                             */
a1=a0                  /**          LIMC                      **/
if mi a1=-a1           /* a1 = |a1|                            */
y=0x3000               /* Set y = 12288                        */
a1-y                   /* |a1| > 12288 ?                       */
if pl a1=y             /* Yes, set |a1| = 12288                */
a0=a0
if mi a1=-a1           /* a1 = A2P                             */
r0=A2P
*r0=a1
r1=TR                  /**          TRIGB                     **/
a0=*r1
a0=a0
if ne a1=a1<<16        /* If TR =| 0, set a1 = 0.              */
r1=A2
*r1=a1
r1=PK1                 /**          UPA1                      **/
r0=PK0
a1=*r1++
y=*r0
a1=a1^y  *r1--=a1      /* a1 = PK0 ^ PK1 = PKS                 */
*r1=y
y=192
move a0=y              /* Set a0 = 192 if PKS = 0              */
if ne a0=-a0           /* otherwise, set a0 = 65344            */
r0=SIGPK
a1=*r0
a1=a1
if ne a0=a0<<16        /* a0=UGA1                              */
r0=A1
y=*r0                  /* y = A1                               */
a0=a0+y                /* a0 = UGA1 + A1                       */
a1=y                   /* a1 = A1                              */
a1=a1>>8               /* a1 = A1 >> 8                         */
y=a1
a0=a0-y                /* a0 = UGA1 + A1 - (A1 >> 8)           */
r2=A1T
y=0x3c00               /**          LIMD                      **/
r1=A2P                 /* Set y = 15360                        */
a1=*r1
a1=a1-y  *r2=a0        /* a1 = A2P - 15360, a0 = A1T           */
a1=-a1
y=a1                   /* y = A1UP                             */
a1=a0
if mi a1=-a1           /* a1 = | A1T |                         */
a1-y                   /* | A1T | - A1UP                      */
if pl a1=y
a0=a0
if mi a1=-a1           /* a1 = A1P                             */
r0=TR                  /**          TRIGB                     **/
a0=*r0
a0=a0                  /* Check TR = 0 ?                       */
if ne a1=a1<<16        /* If not, then set a1 = 0              */
r0=A1
*r0=a1                 /* otherwise, keep a1.                  */
```

```
        r0=DQ0
        y=*r0++
        do 6 {
        *r0zp:y             /* Update the DQn parameters for       */
        }                   /* the next input data.                */
        r0=DQ0S
        y=*r0++
        do 6 {
        *r0zp:y             /* Update the sign bit of DQn for      */
        }                   /* the next input data.                */
        r0=DQCNT0
        y=*r0++
        do 6 {
        *r0zp:y             /* Update the parameters for           */
        }                   /* the next input data.                */
/*********************************************************************/
/*                                                                   */
/*                         TONE                                      */
/*                                                                   */
/*********************************************************************/

totrd1: r0=A2P              /**         TONE  totrde.s            **/
        a0=*r0
        y=0x2e00            /* y = 11776                           */
        a0=a0+y             /* a2(k) + 0.71875                     */
        y=0
        move a1=y
        if mi a1h=a1h+1     /* a1 = TDP                            */
        r0=TDP
        *r0=a1
        r3=TR               /**         TRIGB                     **/
        a0=*r3
        a0=a0               /* Check TR = 0 ?                      */
        if ne a1=a1<<16     /* If not, set a1 = 0                  */
        r1=TD               /* otherwise, keep a1.                 */
        *r1=a1
adspc1: r0=I                /**       FUNCTF  adspc1.s            **/
        a0=*r0              /* a0 = I                              */
        y=FI
        a0=a0+y             /* Use the table looking up to         */
        r2=DML              /* find the FI (scaled) value.         */
        pt=a0
        y=*r2 x=*pt++
        a0=x                /* a0 = FI                             */
        a1=a0<<1            /**         FILTB                     **/
        a1=a1<<1            /* a1 = FI << 11                       */
        a1=a1-y             /* a1 = (FI<<11) - DML = DIF           */
        a1=a1<<1
        a1=a1>>8            /* a1 = DIF >> 7 = DIFSX               */
        a1=a1+y             /* a1 = DIFSX + DML                    */
        y=0x3fff            /* Set y = 16383                       */
        r1=DMS
        a1=a1&y y=*r1
        a0=a0-y *r2=a1      /**         FILTA                     **/
        a0=a0>>1
        a0=a0<<4
        a0=a0>>8            /* a0 = DIFSX = DIF >> 5               */
        a0=a0+y             /* a0 = DIFSX + DMS                    */
        y=0xfff
        a0=a0&y y=*r2
        *r1=a0              /* a0 = DMSP                           */
        a0=a0<<1            /**         SUBTC                     **/
        a0=a0<<1            /* a0 = DMSP << 2                      */
        a0=a0-y  a1=*r2     /* a0 = (DMSP<<2) - DMLP = DIF         */
        if mi a0=-a0        /* a0 = | DIF | = DIFM                 */
```

```
        a1=a1>>4
        a1=a1<<1            /* a1 = DTHR = DMLP >> 3                */
        move y=a1
        r3=Y
        a0=a0-y  a1=*r3     /* a0 = DIFM - DTHR                     */
        r3=TDP
        a0=*r3
        y=1
        if pl a0=y
        y=1536              /* Set y = 1536                         */
        a1=a1-y             /* a1 = Y - 1536                        */
        y=1
        if mi a0=y          /* a0 = AX                              */
        a0=a0<<8            /**         FILTC                    **/
        a0=a0<<1            /* a0 = AX << 9                         */
        r3=AP
        y=*r3               /* y = AP                               */
        a0=a0-y             /* a0 = (AX<<9) - AP = DIF              */
        a0=a0>>4            /* a0 = DIFSX                           */
        a0=a0+y             /* a0 = DIFSX + AP                      */
        y=0x3ff
        r2=TR               /**         TRIGA                    **/
        a0=a0&y y=*r2       /* a0 = APP                             */
        a1=y *r2
        y=0x100             /* Set y = 256                          */
        if ne a0=y
        *r3=a0              /* a0 = APR                             */

/*********************************************************************/
/*                                                                   */
/*                        FUNCTW                                     */
/*                                                                   */
/*********************************************************************/

qsfad1: r0=I                /**         FUNCTW  qsfadp.s         **/
        a0=*r0
        y=WI
        a0=a0+y             /* a0 points to WI                      */
        pt=a0
        y=*r0 x=*pt++       /* x = WI                               */
        a0=x                /**         FILTD                    **/
        a0=a0>>8
        a0=a0>>4
        a0=a0<<1            /* a01 = WI << 5                        */
        r2=Y
        y=0x0
        yl=*r2              /* y = Y                                */
        a0=a0-y             /* a0 = DIF                             */
        a0=a0>>4
        a0=a0>>1            /* a0 = DIFSX = DIF >> 5                */
        a0=a0+y             /* a0 = Y + DIFSX                       */
        a0=a0<<16
        y=0x1fff            /* Set y = 8191                         */
        a0=a0&y             /* a0 = YUT                             */
        y=544               /**         LIMB                     **/
        a0-y                /* If negative, GELL = 1                */
        if mi a0=y
        y=5120              /* Set y = 5120                         */
        a0-y                /* If positive, GEUL = 0                */
        if pl a0=y          /* a0 = YUP                             */
        r3=YU
        *r3=a0              /* Store it.                            */
        y=0
        yl=a0
        r1=YL               /**         FILTE                    **/
        a0=*r1++
```

```
        a0l=*r1--           /* a0 = YL                              */
        a0=-a0              /* a0 = -YL                             */
        a0=a0>>4
        a0=a0>>1
        a0=a0>>1            /* a0 = ( -YL ) >> 6                    */
        a0=a0+y             /* a0 = DIF = DIFSX                     */
        y=*r1++
        yl=*r1--            /* y = YL                               */
        a0=a0+y             /* a0 = YL + DIFSX                      */
        y=0x7
        yl=0xffff
        a0=a0&y             /* a0 = YLP                             */
        *r1++=a0
        *r1=a0l             /* Store it.                            */

/*********************************************************************/
/*                                                                   */
/*                          COMPRESS                                 */
/*                                                                   */
/*********************************************************************/

comprs: r3=temp             /**         COMPRESS  comprs.s         **/
        r0=SR               /* Convert from uniform PCM to          */
        a0=*r0              /* u-law PCM.                           */
        a1=a0>>8
        y=0x80
        a1=a1&y             /* Extract the sign bit.                */
        *r3=a1              /* Store it.                            */
        if ne a0=-a0        /* a0 = IMAG                            */
        y=8158
        a1=a0-y             /* Beyond the border ?                  */
        if gt a0=y          /* Yes, set the maximal value           */
        y=33
        a0=a0+y             /* a0 = a0 + 33                         */
        c1=-10              /* Set the initial counter              */
        do 10 {
        ifc pl a0=a0<<1     /* Find the abc value.                  */
        }
        a0=a0<<1
        a0=a0>>4
        a0=a0>>8
        y=0xf
        a0=a0&y             /* a0 is the xyzw value.                */
        a1=c2
        a1=-a1              /* a1 = abc                             */
        a1=a1<<4
        y=a1
        a0=a0+y  y=*r3      /* y = the sign bit                     */
        a0=a0+y             /* a0 = the before bit inverted         */
        y=0xff
        a0=a0^y             /* a0 = u-law                           */
        r1=S
        *r1=a0              /* Store it.                            */

/*********************************************************************/
/*                                                                   */
/* For the non-standard CCITT ADPCM mode, the program from           */
/* the ipcdsc label, (EXPAND sub-program), to the line               */
/* before the End of Program, (goto loop), will be deleted.          */
/* The output will be stored in RAM location S.                      */
/*                                                                   */
/*********************************************************************/
```

```
/*********************************************************************/
/*                                                                   */
/*                          EXPAND                                   */
/*                                                                   */
/*********************************************************************/

ipcdsc: r1=S                    /**         EXPAND  ipcmv.s          **/
        a0=*r1
        y=0xff                  /* Invert bits.                        */
        a0=a0^y                 /* This program only considers         */
        a1=a0>>4                /* the input as u-law.                 */
        y=0x7
        a1=a1&y                 /* a1 = L = abc                        */
        a1=-a1
        c0=a1
        y=0xf
        a1=a0&y                 /* a1 = V = xyzw                       */
        a1=a1<<1
        y=0x21
        a1=a1|y
        do 7 {
        if c0lt a1=a1<<1
        }
        y=33
        a1=a1-y
        y=0x80                  /* Check the sign bit.                 */
        r0=SE
        a0=a0&y y=*r0
        if ne a1=-a1            /* If negative, then a1 = -a1.         */
        r1=D                    /**         SUBTA                    **/
        a1=a1-y
        *r1=a1

/*********************************************************************/
/*                                                                   */
/*                          LOG                                      */
/*                                                                   */
/*********************************************************************/

adaqu1: r0=D                    /**         LOG  adaqua.s            **/
        y=*r0
        a0=y  *r0
        if mi a0=-a0            /* Convert D from 2's complement.      */
        c1=-15                  /* to signed magnitude                 */
        a1=a0
        do 15 {
        ifc pl a1=a1<<1         /* Compute exponent                    */
        }
        y=0x7f00
        a0=a1&y
        a0=a0>>8
        a1=c2
        y=a0
        a1=-a1
        a1=a1<<8
        a1=a1>>1                /* a1 = EXP << 7                       */
        r0=YSH2
        a1=a1+y  y=*r0          /* y = Y >> 2                          */
        a1=a1-y                 /**         SUBTB                    **/
        y=4095
        a1=a1&y
        r0=DLN                  /* DLN = (DL + 4096 - y) & 4095        */
        *r0=a1
```

```
/*****************************************************************/
/*                                                               */
/*                          SYNC                                 */
/*                                                               */
/*****************************************************************/

sync:   r3=temp1            /**         SYNC  sync.s            **/
        r0=DLN
        a1=*r0              /* a1 = DLNX                        */
        a1=a1>>16           /* a1h = 0x0, a1l = DLNX            */
        r1=0                /* Set counter to find the table    */
        pt=dln              /* offset.                          */
        y=*r3  x=*pt++
ldb3:   p=x*y  y=*r3  x=*pt++    /* Find the table offset       */
        a0=a1-p  *r1++
        if gt goto ldb3
        r0=D
        y=*r0
        a0=y  *r0           /* Check positive or negative ?     */
        y=posx-1
        if pl goto ldb5
        y=negx-1
ldb5:   a0=r1
        a0=a0+y             /* a0 points to the desired DLNX    */
        pt=a0               /* in the table.                    */
        y=*r0  x=*pt++
        r1=S
        a0=*r1              /* a0 = S                           */
        y=0xff
        a0=a0^y             /* Bit inverted                     */
        r0=I
        a1=*r0              /* a1 = I                           */
        y=IM
        a1=a1+y             /* a1 points to the desired IM      */
        pt=a1               /* in the table IM.                 */
        a1=x
        y=a1  x=*pt++
        a1=x                /* a1 = IM                          */
        a1=a1-y             /* Check IM-ID ?                    */
        if eq goto ldb17    /* If zero, SD = SP.                */
        if pl goto ldb11    /* If positive, SD = SP+            */
        y=0x80              /* If negative, SD = SP-            */
        a1=a0&y
        if eq goto ldb8
        y=255
        a1=a0-y
        if ne a0h=a0h+1
        goto ldb17
ldb8:   y=1                 /* The SP- represents the next      */
        a0=a0-y             /* more negative PCM output         */
        y=129               /* level.                           */
        if mi a0=y
        goto ldb17
ldb11:  y=0x80
        a0&y
        if eq goto ldb14
        y=1                  /* The SP+ represents the next     */
        a0=a0-y              /* more positive PCM output        */
        y=127                /* level.                          */
        a0-y
        y=1
        if eq a0=y
        goto ldb17
ldb14:  y=127
        a1=a0-y
```

```
        if ne a0h=a0h+1
ldb17:  y=0xff                  /* Do bit inversion.                  */
        a0=a0^y
        r1=SD
        *r1=a0                  /* Store it.                          */
        goto loop               /* Go back to execute the next        */
                                /* sampling data.                     */

/*********************************************************************/
/*                                                                   */
/*                End of Program                                     */
/*                                                                   */
/*********************************************************************/

posx:   int         0x9
        int         0xa
        int         0xb
        int         0xc
        int         0xd
        int         0xe
        int         0xf
        int         0x7
        int         0x9
negx:   int         0x6
        int         0x5
        int         0x4
        int         0x3
        int         0x2
        int         0x1
        int         0x0
        int         0x7
        int         0x6
WI:     int         0xfff4      /* -12 = 4084 for 12 bits             */
        int         18
        int         41
        int         64
        int         112
        int         198
        int         355
        int         1122
        int         1122
        int         355
        int         198
        int         112
        int         64
        int         41
        int         18
        int         0xfff4
FI:     int         0
        int         0
        int         0
        int         0x200
        int         0x200
        int         0x200
        int         0x600
        int         0xe00
        int         0xe00
        int         0x600
        int         0x200
        int         0x200
        int         0x200
        int         0
        int         0
        int         0
pos:    int         0x01
        int         0x02
```

```
        int     0x03
        int     0x04
        int     0x05
        int     0x06
        int     0x07
        int     0x0f
        int     0x01
neg:    int     0x0e
        int     0x0d
        int     0x0c
        int     0x0b
        int     0x0a
        int     0x09
        int     0x08
        int     0x0f
        int     0x0e
dln:    int     79
        int     177
        int     245
        int     299
        int     348
        int     399
        int     2047
        int     3971
        int     4095
dqln:   int     2048
        int     4
        int     135
        int     213
        int     273
        int     323
        int     373
        int     425
IM:     int     0x8
        int     0x9
        int     0xa
        int     0xb
        int     0xc
        int     0xd
        int     0xe
        int     0xf
        int     0x0
        int     0x1
        int     0x2
        int     0x3
        int     0x4
        int     0x5
        int     0x6
        int     0x7
tabl1:  int     0x0
        int     0x3f
        int     0x3f
        int     0x3f
        int     0x3f
        int     0x3f
        int     0x3f
        int     0x7e
        int     0xfc
        int     0x1f8
        int     0x3f0
        int     0x7e0
        int     0xfc0
        int     0x1f80
        int     0x3f00
        int     0x7e00
```

```
        int         0x7e00
tabl2:  28*int         0x0
        int         0x3
        int         0x7fff
        int         0x6
        int         0x7fff
        int         0xc
        int         0x7fff
        int         0x18
        int         0x7fff
        int         0x30
        int         0x7fff
        int         0x60
        int         0x7fff
        int         0xc0
        int         0x7fff
        int         0x180
        int         0x7fff
        int         0x3
        int         0x6fff
        int         0x6
        int         0x6ffe
        int         0xc
        int         0x6ffc
        int         0x18
        int         0x6ff8
        int         0x30
        int         0x6ff0
        int         0x60
        int         0x6fe0
        int         0xc0
        int         0x6fc0
        int         0x180
        int         0x6f80
        int         0x300
        int         0x6f00

.ram
SD:     int
DLN:    int
S:      int
SL:     int
D:      int
temp:   int
temp1:  int
Y:      int
YSH2:   int
I:      int
R:      int
YL:     int
        int
YU:     int
TDP:    int
AP:     int
DML:    int
DMS:    int
DQ0:    int
DQ1:    int
DQ2:    int
DQ3:    int
DQ4:    int
DQ5:    int
DQ6:    int
SR2:    int
SR1:    int
```

```
B1:      int
B2:      int
B3:      int
B4:      int
B5:      int
B6:      int
A2:      int
A1:      int
         int
WB1:     int
WB2:     int
WB3:     int
WB4:     int
WB5:     int
WB6:     int
WA2:     int
WA1:     int
PK1:     int
PK2:     int
PK0:     int
A2P:     int
SE:      int
SEZ:     int
DQ16:    int        /* The TC of DQ                                    */
TR:      int
A1T:     int
SIGPK:   int
SR:      int
TD:      int
DQ0S:    int
DQ1S:    int
DQ2S:    int
DQ3S:    int
DQ4S:    int
DQ5S:    int
DQ6S:    int
SR2S:    int
SR1S:    int
DQCNT0:  int
DQCNT1:  int
DQCNT2:  int
DQCNT3:  int
DQCNT4:  int
DQCNT5:  int
DQCNT6:  int
SRCNT2:  int
SRCNT1:  int
.endram
```

## References

[1] "Digital Processing of Voice–Band Signals—Algorithm for 24, 32, 40 kbits/s ADPCM," *American National Standard for Telecommunications* (T1Y1/87-040, T1Y1.2/887-090R1).

[2] *WE® DSP16 Digital Signal Processor Information Manual*, June 1987.

[3] "T7500 PCM Codec with Filters" *AT&T Communication Devices Data Book*, March 1988.

For additional information, contact your AT&T Account Manager, or call:

- AT&T Microelectronics
  Dept. 51AL230230
  555 Union Boulevard
  Allentown, PA 18103
  **1-800-372-2447**

  In Canada, call:
  **1-800-553-2448**

- AT&T Microelectronics
  AT&T Deutschland GmbH
  Bahnhofstr. 27A
  D-8043 Unterfoehring
  West Germany
  **Tel. 089/950 86-0**
  **Telefax 089/950 86-111**

- AT&T Microelectronics Asia/Pacific
  4 Shenton Way #20-00
  Shing Kwan House
  Singapore 0106
  **Tel. 225-5233**
  **Telex RS 42898**
  **FAX 225-8725**

- AT&T Microelectronics Japan
  7F, Fukoku Seimei Bldg.,
  2-2-2, Uchisaiwai-cho,
  Chiyoda-ku, Tokyo 100 Japan
  **Tel. (03) 502-3055**
  **Telex J32562 ATTIJ**
  **FAX (03) 593-3307**

- AT&T Microelectronics Spain
  Poligono Industrial de Tres Cantos
  28770 Colmenar Viejo — Madrid
  Spain
  **Tel. (34) 807-1844**
  **FAX (34) 807-1760**

---



October 1988

AP88-07DMOS

AT&T
The right choice.