ANE408

(Previously ANE008)

Logarithmic/Linear Conversion Routines for DSP56000/1

Prepared by:
Eric Cheval
Motorola
Geneva , Switzerland

I. INTRODUCTION

DSP chips are often connected to mono-circuits (cofidecs) in telecommunication applications. When programmed as multichannel DTMF receivers, mail-box vocoders in PBXs, as line equalizers in T1 repeaters , acoustic echo-cancellers or speech scramblers in feature phones, digital signal processors like DSP56000/1 have to exchange 64 kbits/s PCM data with mono-circuits (Fig. 1). These mono-circuits are single-chip 8khz A/D and D/A converters including antialiasing filters (300-3400hz band pass filters) and encoding the analogue samples on 8 bit logarithmic PCM (Pulse Code Modulation) words. These logarithmic coded bytes need to be expanded to 13 or 14 bit linear data words before DSP processing and companded again when output after processing.

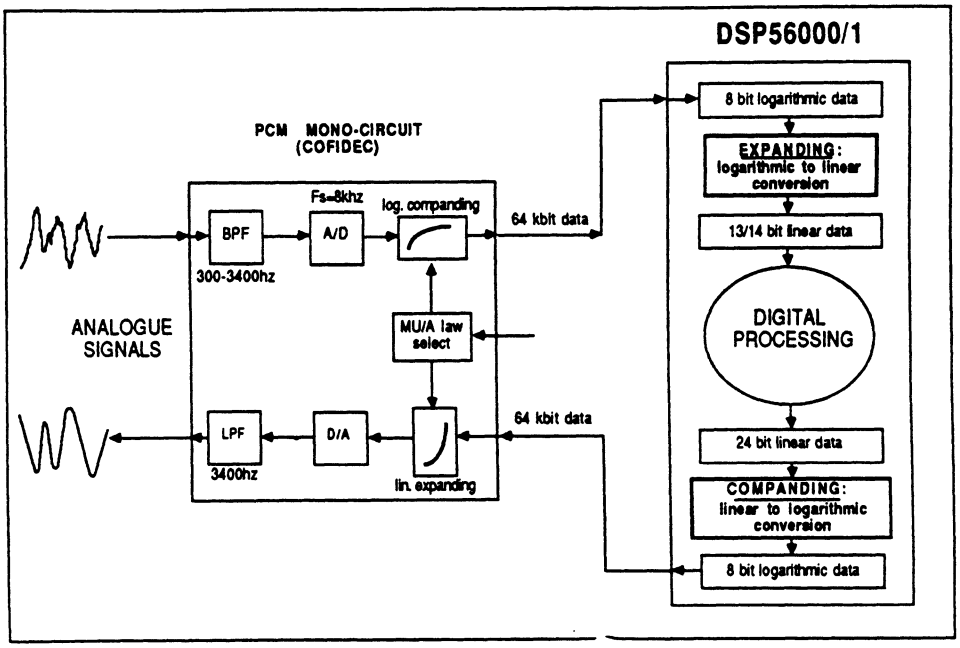

Fig. 1. Logarithmic/Linear conversion

In telephony systems, two companding/expanding laws are used:

-Bell Mu-law for USA/Canada/Japan and the Philippines,

-CCITT A-law for Europe and the rest of the world.

Conversion techniques differ only on some points for the BELL and the CCITT standards and most existing mono-circuits are able to handle the two laws .The Synchronous Serial Interface (SSI) of DSP56000/1 makes this chip very suitable for any application which requires digital processing of PCM data coming from one device(SSI normal mode) or coming from a standard 24 slot BELL or 32 slot CCITT TDM (time division multiplex) system. The SSI network mode provides the software control necessary to handle between 2 and 32 slots without any glue logic, the time slot assignment (TSA) being totally under software control.

Section II of this note gives the definition of the two laws as well as some decoding/encoding examples after an introduction on logarithmic quantisation. Routine flowcharts and listings are found in section III and the appendices of section IV contain the decoding tables as well as the fortran routines used to generate the tables.

II. THEORY AND DEFINITION

Digitalisation of a continuous analogue signal into a discrete sequence of digital characters is carried out using two main steps:
SAMPLING: the continuous signal x(t) is transformed into a discrete sequence x(kT) where x(kT)=x(t) for t=kT . Fs=1/T is the sampling frequency which has to satisfy the Shannon theorem:

$$Fs > 2 \times B$$

where B is the bandwidth of the incident signal. The sampling process is equivalent to amplitude modulation of a constant amplitude pulse sequence (Fig. 2) and the sampled signal is generally referred as a PAM (Pulse Amplitude Modulation) signal.

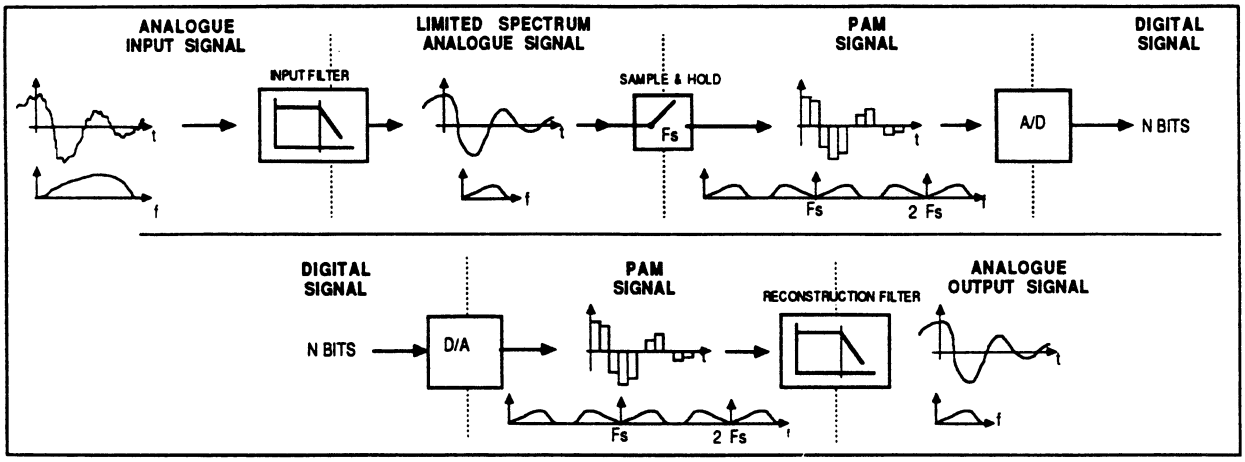

Fig. 2. Digital/Analogue conversion

In our case,the bandwidth of the telephone channel is limited to frequencies in the band 300-3400 Hz. This bandwidth limitation allows a sampling rate of Fs = 8Khz. Antialiasing and reconstruction filters specified by the CCITT will remove any signal components outside this frequency band.
QUANTISATION: it is the basic operation of digitalisation. The PAM input signal is coded using a discrete set of signal level leading to a digital word. This transformation is not linear and involves a loss of information: after reconstruction, the signal will differ from the incident signal. This error is called the **"quantisation noise"** and is shown in Fig. 3.

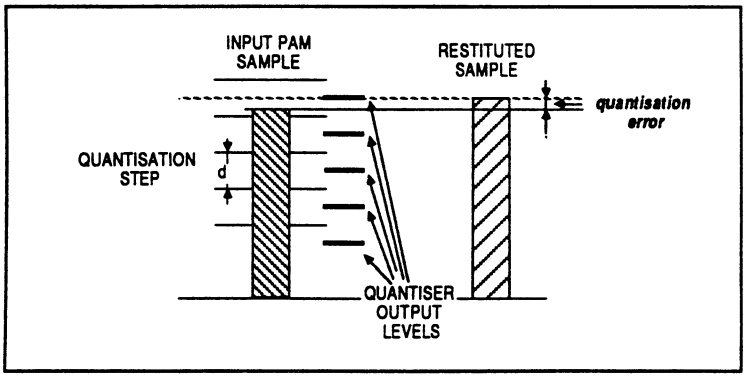

Fig. 3. Quantisation error

The power of the quantisation noise depends mainly on three parameters:

- -number of quantisation levels q, directly related with the number of bits n on which the sample will be coded($n= \log_2 (q)$),
- -the quantisation step d which can be uniform or not (when steps are uniform, the same quantisation interval length is used for all samples and the quantisation is said to be uniform or linear),
- -the probability density p(x) of the incident signal in each quantisation segment.

The quantisation noise reduction problem can be formulated as follows: given a number of bits n (or a number of quantisation levels) , and given a probability density p(x), the quantisation function has to optimize two criteria:

a) minimise quantisation noise,

b) maximise information quantity at the quantiser output (all levels equally probable).

Here the quantiser is chosen considering some constraints for digital speech on the telecommunication network:

- **65 dB dynamic range**.

This fixes the number of levels on which the linear quantisation will be done. If n is the number of bits selected, the relation is:

$$65\,dB < 20 \log \frac{2Vm}{2^n}$$

where [-Vm,+Vm] is the signal range. If Vm=1 (normalized signal), we get:

$$n-1 > 10.8$$

which means that at least 12 bits are necessary.

- **more than 35 dB signal to noise ratio (S/N) in a large range of signal amplitudes (around 30 dB)**.

The signal to noise ratio for a linear quantisation of n bits is given by:

$$S/N = 3q^2 \frac{Si^2}{Vm^2}$$

where $q = 2^n - 1$ and Si is the signal energy. Expressed in dB gives:

$$(S/N)db = 10\log 3 + 20\log q + (Si)dB.$$

Taking a mean value (standard speaker) for (Si)dB like -15 dBm0[1], we obtain:

$$35\,dB < 10\log 3 + 20\log(2^n - 1) - 15$$

which makes n >12.

These two speech intelligibility constraints lead to at least a 12 bit word when linear quantisation is used. But this solution can not be selected because the bit rate of a single digital voice channel has to be as low as possible in order to transmit the maximum number of channel on the same medium using Time Division Multiplex (TDM) techniques. 96 Kbit/s, the bit rate of 12 bit data sampled at 8 Khz, is too high.

On the other hand, an 8 bit linear quantiser is not satisfactory for two reasons:

-the dynamic range is under 65 dB,

-the signal-to-noise ratio of a linear quantiser varies linearly with the signal energy and this is not appropriate when a constant ratio is requested for a wide dynamic input signal . An 8 bit linear quantiser S/N ratio is not always greater than 35 dB in the signal level range [-30,0]dBm0.

In conclusion, a 12 bit linear quantiser gives a bit rate too high and a 8 bit linear quantiser has too low a S/N ratio for low amplitudes.

Given these two considerations,an 8 bit non-uniform quantisation has been defined. It can be proved that the signal to quantisation-noise becomes independent from the signal level if the quantisation function is a logarithmic function like:

$$y = 1 + \frac{\log x}{k}$$

for x >0 , and the symmetric curve for x<0 . Using such a function, the size of quantisation step increases with the input signal value: small amplitudes are coded with finer granularity (more steps) than larger amplitudes. S/N ratio becomes constant because each signal level is coded with the same relative precision (see Fig. 4). In practice , this function can not be used for small signal levels because when x -> 0 , log x becomes infinite. An approximation of this curve will be necessary for small levels.

---

[1]- 0 dBm0 = a signal of 1 mW in a 600 ohm resistor.

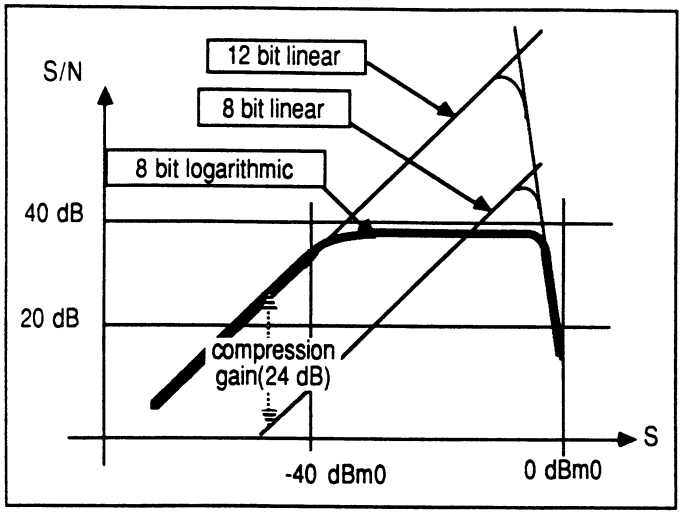

Fig. 4. Logarithmic compression Signal-to-Noise ratio

The chosen 8 bit logarithmic coding laws are implemented in two steps: first a 13/14 bit linear quantisation and second a logarithmic compression to 8 bits. Of course the reverse operations will be applied for the decoding phase. The approximation around the origin is different for A and Mu laws (see Fig. 5 & Table 1):

-for the A-law, the curve is approximated with its tangent near the origin, so that quantisation is linear for small signals. The slope of the linear part, called the compression rate, has been fixed to 16 and is given by:

$$C = \frac{A}{1 + \ln A} = 16$$

which makes A=87.6.

-for the Mu-law, the function taken is quasi-linear for x small and quasi-logarithmic for x large. The compression rate near the origin is given by:

$$C = \frac{Mu}{\ln(1 + Mu)}$$

Mu = 255 gives C=46

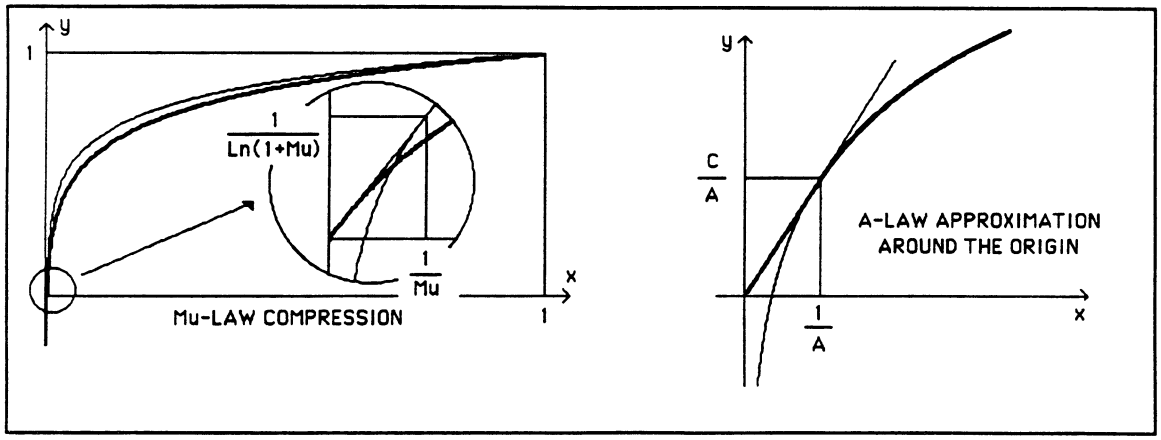

Fig. 5. A & MU laws approximation around the origin

A-LAW

$$F(x) = Sgn(x)\left[\frac{1 + \ln(A|x|)}{1 + \ln A}\right], \quad \frac{1}{A} < |x| < 1$$

$$= Sgn(x)\left[\frac{A|x|}{1 + \ln A}\right], \quad 0 < |x| < \frac{1}{A}$$

MU-LAW

$$F(x) = Sgn(x)\,\frac{\ln(1 + Mu\,|x|)}{\ln(1 + Mu)}, \quad 0 < |x| < 1$$

where
- x = input signal
- $Sgn(x)$ = sign of the input x
- $|x|$ = absolute value of x (magnitude)
- MU = 255 (defined by AT&T)
- A = 87.6 (defined by CCITT)

Table 1. MU & A law encoding functions

Mathematically speaking, MU & A-laws are piece-wise linear approximations of the above equations. The two laws are very similar, they both have a quasi-constant signal to noise ratio around 38 dB for medium level signals [-30, 0 dbm0]. However, the difference around the origin gives the MU-law a better signal to noise ratio for very low level signals. Details on conversion between the two laws will be found in [5] . The 8 bit digital code has a sign bit P, three bits of segment S2S1S0, and 4 bits Q3Q2Q1Q0 for step selection within the chosen segment.

BIT	8	7	6	5	4	3	2	1
	P	S2	S1	S0	Q3	Q2	Q1	Q0

The way of coding the byte is different for the two laws. For A-law,the sign bit is 1 for positive values and even bit are inverted before transmission (bit 2,4,6,8 when sign bit is bit 8). For MU-law, the sign bit is 1 for negative values and the byte is negated before transmission. S2S1S0 is the segment code. For the two values of the sign bit P this makes 16 segments,but the segments cutting the origin are collinear and counted as 1. In the A-law, the two segments near the origin have the same slope for positive and negative number, so that this law has in fact 13 different segments (four segments counted as one). The MU-law is a 15 segment law because the first positive and the first negative segment count as one. Q3Q2Q1Q0 are the 4 bits used for uniform quantisation of 16 levels in each segment. Another difference between MU-law and A-law is the decoder output characteristic around zero (Fig. 6). The MU-law has a dead zone around zero which requires two codes (plus and minus zero). The A-law coder has a threshold at zero: for zero input, the output has an offset of plus or minus 1/2 step:

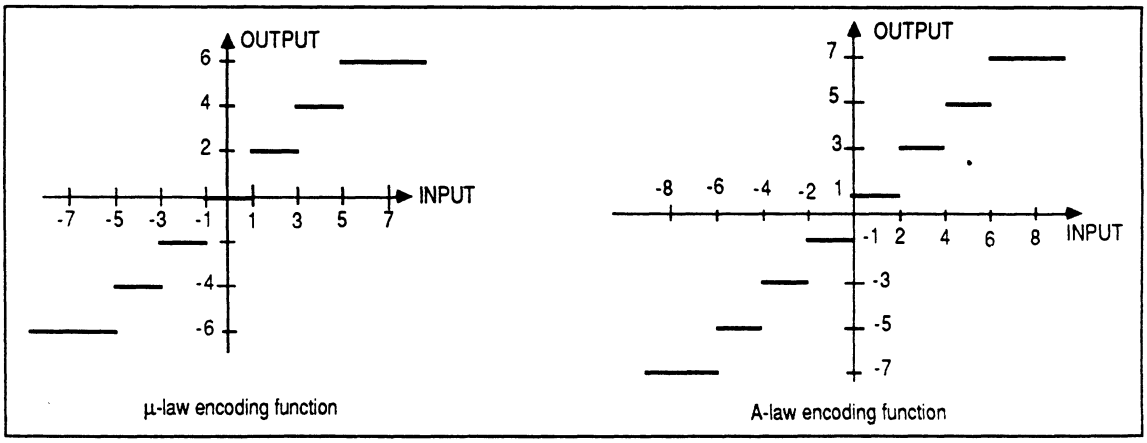

Fig. 6 . Encoding function around the origin

Fig. 7 describes the A-LAW quantisation and coding process. Even bits are inverted before transmission and for negative values, the sign bit is zero.

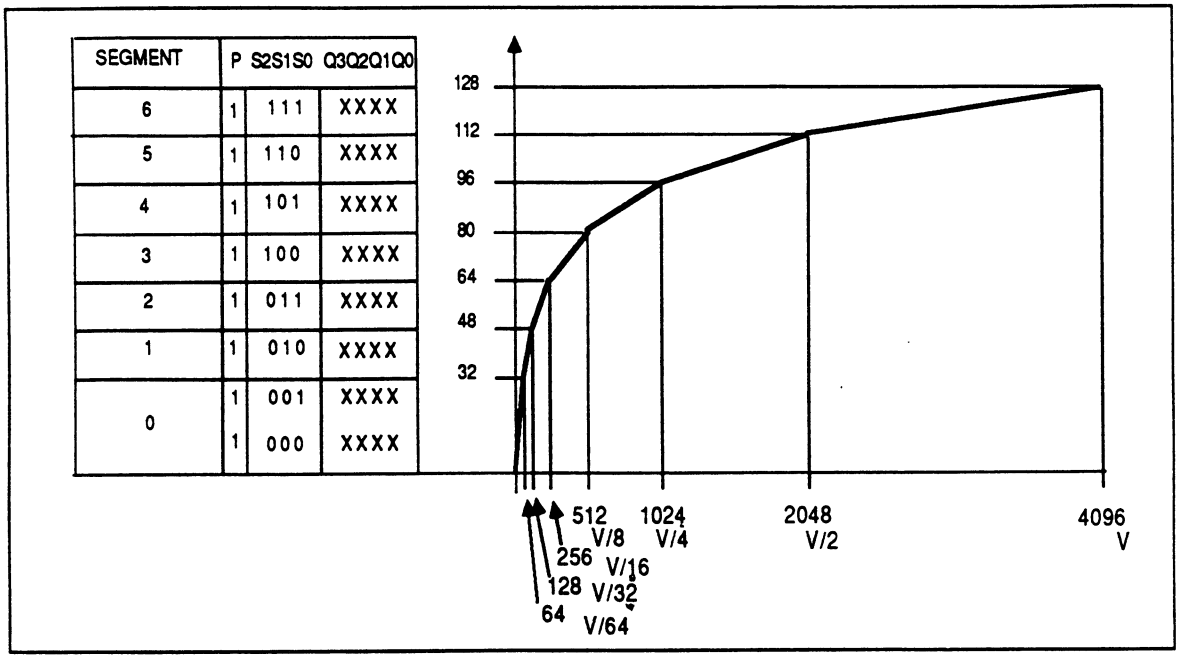

Fig. 7. Quantisation and coding for the A-law

The A-law coding/decoding table for positive and negative numbers (Tables 2 & 3) are given on the following pages with two decoding/encoding examples. The A coding law is:

Positive values

- 127 decision values are used. n=128 is a virtual decision,
- x between decision values [x(n), x(n+1)] is coded as 128 + n before the even bit inversion,
- the decoder output value will then be :

$$y(n) = \frac{x(n-1) + x(n)}{2} \quad \text{for } n=1,\ldots,128$$

Negative values

- 127 decision values are used. n=128 is a virtual decision
- x between decision values [x(n), x(n+1)] is coded as n before the even bit inversion,
- the decoder output value will then be :

$$y(n) = \frac{x(n-1) + x(n)}{2} \quad \text{for } n=1,\ldots,128$$

Segment Number S2S1S0	Number of intervals and size	Decision value number	Value at segment end points	Coded byte before even bit inversion 12345678	hex	Value at decoder output
		(128)	(4096)			
		127	3968	11111111	FF	4032
				11111110	FE	3904
		126	3840	11111101	FD	3776
7 111	16 x 128	|	|	|	|	|
		113	2176			
		112	2048	11110000	F0	2112
				11101111	EF	2016
		111	1884			
6 110	16 x 64	|	|	|	|	|
		97	1088			
		96	1024	11100000	E0	1056
				11011111	DF	1008
		95	992			
5 101	16 x 32	|	|	|	|	|
		81	544			
		80	512	11010000	D0	528
				11001111	CF	504
		79	496	11001110	CE	488
4 100	16 x 16	|	|	|	|	|
		65	272			
		64	256	11000000	C0	264
				10111111	BF	252
		63	248			
3 011	16 x 8	|	|	|	|	|
		49	136			
		48	128	10110000	B0	132
				10101111	AF	126
		47	124			
2 010	16 x 4	|	|	|	|	|
		33	68			
		32	64	10100000	A0	66
1 001				10011111	9F	63
		31	62			
	32 x 2	|	|	|	|	|
		2	4			
				10000001	81	3
0 000 (cont.)		1	2			
				10000000	80	1
		0	0			

Table 2. A-law positive value coding/decoding table

Segment Number S2S1S0	Number of intervals and size	Decision value number	Value at segment end points	Coded byte before even bit inversion 12345678	hex	Value at decoder output
(cont.)		0	0			
0 000		1	-2	00000000	0	1
				00000001	1	-3
		2	-4			|
	32 x 2	|	|	|	|	
1 001		31	-62			|
				00011111	1F	-63
		32	-64	00100000	20	-66
		33	-68	00100001	21	-70
		34	-72			|
2 010	16 x 4	|	|	|	|	
		47	-124			|
				0010FFFF	2F	-126
		48	-128	00110000	30	-132
		49	-136			|
3 011	16 x 8	|	|	|	|	
		63	-248			|
				00111111	3F	-252
		64	-256	01000000	40	-264
		65	-272	01000001	41	-280
4 100	16 x 16	|	|	|	|	|
		79	-496			
				0100FFFF	4F	-504
		80	-512	01010000	50	-528
		81	-544			
5 101	16 x 32	|	|	|	|	|
		95	-992			
				0101FFFF	5F	-1008
		96	-1024	01100000	60	-1056
		97	-1088			
6 110	16 x 64	|	|	|	|	|
		111	-1884			
				0110FFFF	6F	-2016
		112	-2048	01110000	70	-2112
		113	-2176			
7 111	16 x 128	|	|	|	|	|
				01111110	7E	-3904
		127	-3968	01111111	7F	-4032
		(128)	(-4096)			

Table 3. A-law negative value coding/decoding table

logarithmic input data							linear output data										
22	21	20	19	17	16	15	22	21	20	19	18	17	16	15	14	13	12	11
6	5	4	3	2	1	0	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	Q3	Q2	Q1	Q0	0	0	0	0	0	0	0	Q3	Q2	Q1	Q0	1
0	0	1	Q3	Q2	Q1	Q0	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	1
0	1	0	Q3	Q2	Q1	Q0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	1	0
0	1	1	Q3	Q2	Q1	Q0	0	0	0	0	1	Q3	Q2	Q1	Q0	1	0	0
1	0	0	Q3	Q2	Q1	Q0	0	0	0	1	Q3	Q2	Q1	Q0	1	0	0	0
1	0	1	Q3	Q2	Q1	Q0	0	0	1	Q3	Q2	Q1	Q0	1	0	0	0	0
1	1	0	Q3	Q2	Q1	Q0	0	1	Q3	Q2	Q1	Q0	1	0	0	0	0	0
1	1	1	Q3	Q2	Q1	Q0	1	Q3	Q2	Q1	Q0	1	0	0	0	0	0	0

Table 4. A-LAW decoding table

DECODING EXAMPLES:

- the PCM byte B2 hex is decoded as 2F0000 hex:

```
      P S2 S1 S0 Q3 Q2 Q1 Q0
bit 23 22 21 20 19 18 17 16 15 14 13 12 11
     1| 0  1  1| 0  0  1  0|                 even bits are restored
     0  1  0  1  0  1  0  1                   by XOR with 55 hex
    ------------------------                  it gives:
     1| 1  1  0| 0  1  1  1|                  which corresponds to:
     0  0  1  0  1  1  1  1  0  0  0  0  0 according to the
                                               table and the sign
                                               bit inversion.
```

- the PCM byte 46 hex is decoded as FEC800 hex:

```
      P S2 S1 S0 Q3 Q2 Q1 Q0
bit 23 22 21 20 19 18 17 16 15 14 13 12 11
     0| 1  0  0| 0  1  1  0|                 even bits are restored
     0  1  0  1  0  1  0  1                   by XOR with 55 hex.
    ------------------------                  it gives:
     0| 0  0  1| 0  0  1  1|                  which corresponds to:
     0  0  0  0  0  0  0  1  0  0  1  1  1 according to the law.
     1  1  1  1  1  1  1  0  1  1  0  0  1 is the complement
                                               value (because P=0)
```

linear input data												logarith. output data						
[2] [3]	22 11	21 10	20 9	19 8	18 7	17 6	16 5	15 4	14 3	13 2	12 1	11 0	22 6	21 5	20 4	19 3	18 2	17 1	16 0
	0	0	0	0	0	0	0	Q3	Q2	Q1	Q0	X	0	0	0	Q3	Q2	Q1	Q0
	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	X	0	0	1	Q3	Q2	Q1	Q0
	0	0	0	0	0	1	Q3	Q2	Q1	Q0	X	X	0	1	0	Q3	Q2	Q1	Q0
	0	0	0	0	1	Q3	Q2	Q1	Q0	X	X	X	0	1	1	Q3	Q2	Q1	Q0
	0	0	0	1	Q3	Q2	Q1	Q0	X	X	X	X	1	0	0	Q3	Q2	Q1	Q0
	0	0	1	Q3	Q2	Q1	Q0	X	X	X	X	X	1	0	1	Q3	Q2	Q1	Q0
	0	1	Q3	Q2	Q1	Q0	X	X	X	X	X	X	1	1	0	Q3	Q2	Q1	Q0
	1	Q3	Q2	Q1	Q0	X	X	X	X	X	X	X	1	1	1	Q3	Q2	Q1	Q0

Table 5 . A-LAW encoding table

CODING EXAMPLES:

- FA2400 hex linear is coded as 37 and transmitted as 62 hex:

```
bit 23  22  21  20  19  18  17  16  15  14  13  12  11
     1   1   1   1   1   0   1   0   0   0   1   0   0 which negated gives:
     0   0   0   0   0   1   0   1   1   1   0   1   1 which corresponds to:
      | 0   1   1 |0   1   1   1|           (according to the table)

FA2400 is negative (P=0) and will be coded :
P|S2S1S0|Q3Q2Q1Q0 =0|011|0111 which is 37 hexadecimal.  Even
bits are inverted before transmission doing a XOR by 55
hexadecimal which gives 62 hexadecimal.
```

- 0E3C00 hex. linear is coded as CC hex and transmitted 99 hex:

```
bit 23  22  21  20  19  18  17  16  15  14  13  12  11
     0   0   0   0   1   1   1   0   0   1   1   1   1 which corresponds to:
      | 1   0   0 |1   1   0   0|           (according to the table)

0E3C00 is positive (P=1) and will be coded :
P|S2S1S0|Q3Q2Q1Q0 =1|100|1100 which is CC hexadecimal.  Even
bits are inverted before transmission doing a XOR by 55
hexadecimal which gives 99 hexadecimal.
```

---

[2] bit number in DSP56000/1 (bit 23 is sign bit)

[3] bit number in the data (bit 12 is sign bit)

Fig. 8 describes the MU-LAW quantisation and coding process. All bits are inverted before transmission and for negative values, the sign bit is one.

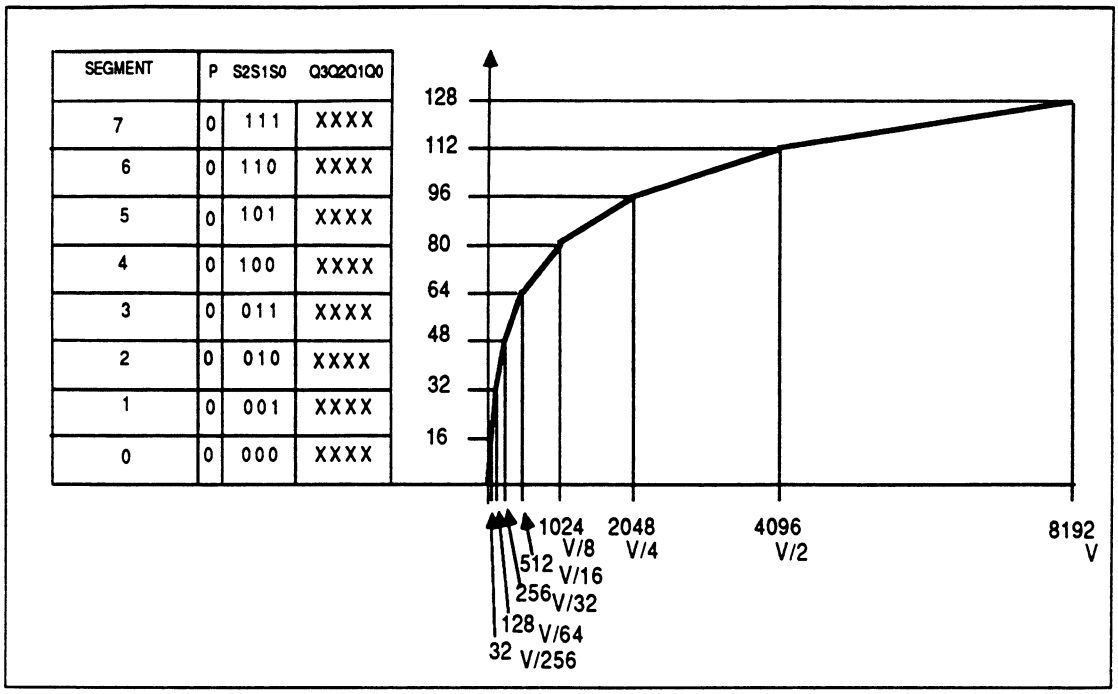

Fig. 8 . Quantisation and coding for the MU-law

The MU-law coding/decoding table for positive and negative numbers (Tables 6 & 7) are given on the following pages with two decoding/encoding examples. The MU coding law is:

Positive values:

- 127 decision values are used. n=128 is a virtual decision,
- x between decision values [x(n), x(n+1)] is coded and transmitted as 255 - n,
- the decoder output value will then be :

$$y(n) = \frac{x(n) + x(n+1)}{2} \quad \text{for } n=1,\ldots,127$$

$$y(0) = 0$$

Negative values:

- 127 decision values are used. n=128 is a virtual decision,
- x between decision values [x(n), x(n+1)] is coded and transmitted as 127 - n,
- the decoder output value will then be :

$$y(n) = \frac{x(n) + x(n+1)}{2} \quad \text{for } n=1,\ldots,127$$

$$y(0) = 0$$

Segment Number S2S1S0	Number of intervals and size	Decision value number	Value at segment end points	Coded byte before all bit inversion 12345678	hex	Value at decoder output
		(128)	(8159)			
				10000000	80	8031
		127	7903			
7 111	16 x 256	|	|	10000001	81	7775
		113	4319	|	|	|
				10001111	8F	4191
		112	4063	10010000	90	3999
		111	3935			|
6 110	16 x 128	|	|	|	|	
		97	2143			|
				10011111	9F	2079
		96	2015	10100000	A0	1983
		95	1951			|
5 101	16 x 64	|	|	|	|	|
		81	1055			
				1010FFFF	AF	1023
		80	991	10110000	B0	975
		79	959			
4 100	16 x 32	|	|	|	|	|
		65	511			
				1011FFFF	BF	495
		64	479	11000000	C0	471
		63	463			
3 011	16 x 16	|	|	|	|	|
		49	239			
				11001111	CF	231
		48	223	11010000	D0	219
		47	215			
2 010	16 x 8	|	|	|	|	|
		33	103			
				1101FFFF	DF	99
		32	95	11100000	E0	93
		31	91			
1 001	16 x 4	|	|	|	|	|
		17	35			
				1110FFFF	EF	33
		16	31	11110000	F0	30
		15	29			
0 000	15 x 2	|	|	|	|	|
		2	3			
				11111110	FE	2
(cont.)		1	1			
	1 x 1			11111111	FF	0
		0	0			

Table 6. MU-law positive value coding/decoding table

Segment Number S2S1S0	Number of intervals and size	Decision value number	Value at segment end points	Coded byte before all bit inversion 12345678	hex	Value at decoder output
(cont.)		0	0			
	1 x 1	1	1	01111111	7F	0
0 000				01111110	7E	-2
		2	3			|
	15 x 2	|	|	|	|	|
		15	-29			|
		16	-31	01110000	70	-30
				01101111	6F	-33
		17	-35			|
1 001	16 x 4			|	|	
		31	-91			|
		32	-95	01100000	60	-93
				01011111	5F	-99
		33	-103			|
2 010	16 x 8	|	|	|	|	
		47	-215			|
		48	-223	01010000	50	-219
				01001111	4F	-231
		49	-239			|
3 011	16 x 16	|	|	|	|	|
		63	-463			
		64	-479	01000000	40	-471
				00111111	3F	-495
		65	-511	00111110	3E	-527
4 100	16 x 32	|	|	|	|	|
		79	-959			
		80	-991	00110000	30	-975
				00101111	2F	-1023
		81	-1055			
5 101	16 x 64	|	|	|	|	|
		95	-1951			
		96	-2015	00100000	20	-1983
				00011111	1F	-2079
		97	-2143			
6 110	16 x 128	|	|	|	|	|
		111	-3935			
		112	-4063	00010000	10	-3999
				00001111	0F	-4191
		113	-4319			
7 111	16 x 256	|	|	|	|	|
		127	-7903	00000001	01	-7775
				00000000	00	-8031
		(128)	(-8159)			

Table 7. MU-law negative value coding/decoding table

logarithmic input data							linear biased output data											
22 6	21 5	20 4	19 3	17 2	16 1	15 0	22 12	21 11	20 10	19 9	18 8	17 7	16 6	15 5	14 4	13 3	12 2	11 1	10 0
0	0	0	Q3	Q2	Q1	Q0	0	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	1
0	0	1	Q3	Q2	Q1	Q0	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	1	0
0	1	0	Q3	Q2	Q1	Q0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	1	0	0
0	1	1	Q3	Q2	Q1	Q0	0	0	0	0	1	Q3	Q2	Q1	Q0	1	0	0	0
1	0	0	Q3	Q2	Q1	Q0	0	0	0	1	Q3	Q2	Q1	Q0	1	0	0	0	0
1	0	1	Q3	Q2	Q1	Q0	0	0	1	Q3	Q2	Q1	Q0	1	0	0	0	0	0
1	1	0	Q3	Q2	Q1	Q0	0	1	Q3	Q2	Q1	Q0	1	0	0	0	0	0	0
1	1	1	Q3	Q2	Q1	Q0	1	Q3	Q2	Q1	Q0	1	0	0	0	0	0	0	0

Table 8. MU-LAW decoding table

DECODING EXAMPLES:

- the PCM byte B2 hex is decoded as 0E3C00 hex :

```
      P S2 S1 S0 Q3 Q2 Q1 Q0
bit 23 22 21 20 19 18 17 16 15 14 13 12 11 10
     1| 0  1  1| 0  0  1  0|                    all bit inversion:
     0| 1  0  0| 1  1  0  1|                    corresponds to:
     0  0  0  0  1  1  1  0  1  1  0  0  0  0    0EC000
     0  0  0  0  0  0  0  0  1  0  0  0  0  1   -008400 (bias)
    ---------------------------------------- --------
     0  0  0  0  1  1  1  0  0  0  1  1  0  0    0E3C00
```

- the PCM byte 46 hex is decoded as FA2400 hex:

```
      P S2 S1 S0 Q3 Q2 Q1 Q0
bit 23 22 21 20 19 18 17 16 15 14 13 12 11 10
     0| 1  0  0| 0  1  1  0|                    all bit inversion:
     1| 0  1  1| 1  0  0  1|                    corresponds to:
     0  0  0  0  0  1  1  0  0  1  1  0  0  0    066000
     0  0  0  0  0  0  0  0  1  0  0  0  0  1   -008400 (bias)
    ---------------------------------------- --------
     0  0  0  0  0  1  0  1  1  1  0  1  1  1    05DC00

 which has to be inverted:
     1  1  1  1  1  0  1  0  0  0  1  0  0  1    FA2400
```

linear biased input data													logarithmic output data						
2 3	22 12	21 11	20 10	19 9	18 8	17 7	16 6	15 5	14 4	13 3	12 2	11 1	10 0	22 6	21 5	20 4	19 3	18 2	17 1	16 0
	0	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	X	0	0	0	Q3	Q2	Q1	Q0
	0	0	0	0	0	0	1	Q3	Q2	Q1	Q0	X	X	0	0	1	Q3	Q2	Q1	Q0
	0	0	0	0	0	1	Q3	Q2	Q1	Q0	X	X	X	0	1	0	Q3	Q2	Q1	Q0
	0	0	0	0	1	Q3	Q2	Q1	Q0	X	X	X	X	0	1	1	Q3	Q2	Q1	Q0
	0	0	0	1	Q3	Q2	Q1	Q0	X	X	X	X	X	1	0	0	Q3	Q2	Q1	Q0
	0	0	1	Q3	Q2	Q1	Q0	X	X	X	X	X	X	1	0	1	Q3	Q2	Q1	Q0
	0	1	Q3	Q2	Q1	Q0	X	X	X	X	X	X	X	1	1	0	Q3	Q2	Q1	Q0
	1	Q3	Q2	Q1	Q0	X	X	X	X	X	X	X	X	1	1	1	Q3	Q2	Q1	Q0

Table 9. MU-LAW encoding table

CODING EXAMPLES:

- FA2400 hex linear is coded B9 and transmitted as 46 hex :

```
bit 23 22 21 20 19 18 17 16 15 14 13 12 11 10
     1  1  1  1  1  0  1  0  0  0  1  0  0  0  negated gives:
     0  0  0  0  0  1  0  1  1  1  0  1  1  1  which biased by
     0  0  0  0  0  0  0  0  1  0  0  0  0  1  0084 hexadecimal
      ---------------------------------------- gives:
     0  0  0  0  0  1  1  0  0  1  1  0  0  0  it corresponds to:
      | 0  1  1 |1  0  0  1|            (according to the table)
```

FA2400 is negative (P=1) and will be coded :
P|S3S2S1|Q3Q2Q1Q0 =1|011|1001 which is B9 hexadecimal. All bits are inverted before transmission giving 46 hexadecimal.

- 0E3C00 hex linear is coded 4D and transmitted as B2 hex:

```
bit 23 22 21 20 19 18 17 16 15 14 13 12 11 10
     0  0  0  0| 1  1  1  0| 0  0  1  1| 1  1  which biased by
     0  0  0  0  0  0  0  0  1  0  0  0  0  1  0084 hexadecimal
      ---------------------------------------- gives:
     0  0  0  0  1  1  1  0  1  1  0  0  0  0  it corresponds to:
      | 1  0  0 |1  1  0  1 |            (according to the table)
```

0E3C00 is positive (P=0) and will be coded :
P|S3S2S1|Q3Q2Q1Q0 =0|100|1101 which is 4D hexadecimal. All bits are inverted before transmission giving B2 hexadecimal.

---

[2] bit number in DSP56000/1 (bit 23 is sign bit)

[3] bit number in the data (bit 12 is sign bit)

1) TECHNIQUES and PROGRAMS

Expanding 8 bit logarithmic data or companding linear data to a byte will result in a certain amount of program cycles and data memory words. In some cases the programmer will have to convert very quickly without taking care of the necessary data memory size. In other cases, he will try to minimize the requested data memory , even if the conversion becomes slower. In order to offer a wide conversion choice, several solutions are given in the following pages. Three expansion techniques are proposed. The first two minimize the program memory size and the execution time doing the conversion with tabulated data. They require a large data memory buffer to store the table (256 words for one , 128 for the other) and a very small program. The third one is not using any tabulated data but request more instructions. Two versions for companding are described . The first one does not use any data memory locations, and the second one, much faster, uses 8 data memory locations.

The respective programs are listed on the following pages and are:

- for the A-law:
  - **ALGLIN1.ASM:** A-law log/lin. conversion using 256 data words
  - **ALGLN11.ASM:** A-law log/lin. conversion using 128 data words
  - **ALGLIN2.ASM:** A-law log/lin. conversion using 1 data words
  - **ALINLG1.ASM:** A-law lin/log. conversion using 0 data word
  - **ALINLG2.ASM:** A-law lin/log. conversion using 8 data words
- for the Mu-law:
  - **MLGLIN1.ASM:** Mu-law log/lin. conversion using 256 data words
  - **MLGLN11.ASM:** Mu-law log/lin. conversion using 128 data words
  - **MLGLIN2.ASM:** Mu-law log/lin. conversion using 1 data word
  - **MLINLG1.ASM:** Mu-law lin/log. conversion using 0 data word
  - **MLINLG2.ASM:** Mu-law lin/log. conversion using 9 data words

Any of the expanding algorithms can be used with any of the companding ones. The expansion tables used by MLGLIN1.ASM and ALGLIN1.ASM have been put in the macros MLOGLIN (file MLOGLIN.ASM)and ALOGLIN (file ALOGLIN.ASM). Tables used by MLGLN11.ASM and ALGLN11.ASM are not called as a macro and are supposed to be present in the internal memory locations X:$180-$1ff for the Mu-law and Y:$180-$1ff for the A-Law. All these tables are listed in the appendices. All programs using a macro assume that this macro is located in the MS-DOS directory '\dsp56000.mac\' and this directory is given as the operand of the MACLIB directive (see ASM56000 user manual).

2) A-LAW CONVERSION ROUTINES

For each program, a flowchart given before the source listing will help the programer to understand the routines. Tables ALOGLIN and ALGLNXP used by ALGLIN1 and ALGLN11 will be found respectively in appendices A & B. A Fortran routine used to generate the table contained in ALGLNXP.ASM is given appendix D. This routine can be modified in order to generate the complete natural order table ALOGLIN.ASM by changing three instructions:

-avoid the CALL SCRAMBLE,
-change "DO 20 I=129,256" to "DO 20 I=1,256"
-insert a macro statement in WRITE 999.

The following table gives an overview of the performance and requirements of each technique (time information is given for Fosc=20.5Mhz):

Function	Routine	Memory Size Prog	Memory Size Data	Cycles	Time (micro s)
Expand	ALGLIN1	11	256	12	1.17
	ALGLN11	15	128	18	1.76
	ALGLIN2	28	1	34	3.32
Compress	ALINLG1	18	0	44	4.29
	ALINLG2	16	8	26	2.54

Table 10 : Memory Space and Time Requirements for A-Law Conversion

2-a) A-law decoding routines.

ALGLIN1.ASM & ALGLN11.ASM

The principle of these two routines is very simple. The PCM byte to be decoded is used to address a table in which the corresponding linear value has been stored.

ALGLIN1.ASM has to invert the even bit before addressing because the 256 element table has been organized in the natural order. The flowchart of this routine is given in Fig. 9.

ALGLN11.ASM is working with a 128 element table in the even bit inverted order: no even bit inversion is necessary but a small process is necessary for the sign (table contains only positive values). This routine can use the table programmed in DSP56001 X data ROM at starting address X:$180.

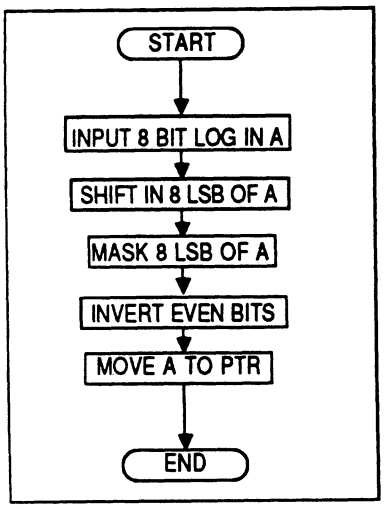

Fig. 9 . ALGLIN1.ASM flowchart

```
        page    75,72,0,10
        opt     nomd,nomex
;
;ALGLIN1.ASM
;
;A-law to linear conversion using a table of data
;
;    the 8 bit log data to be decoded
;    is in the ssi txrx register highbyte (bit23-bit15)
;       r3 points to the 13 bit linear data
;
; timing in cycles: 12 (-> 1.17 micro s)
; required memory space: - 256 data memory word
;                        -  11 program memory words
;
start   equ     $40             ;program starting address
tab     equ     0               ;table starting address
shift   equ     $80             ;used for 16 bit asr
mask    equ     $ff             ;pcm byte mask
inv1    equ     $000055         ;invert even bit of 8 lsb
txrx    equ     $ffef           ;SSI rx register address
;
        maclib  '\dsp56000.mac\'    ;MS-DOS macro dir.
;
        aloglin tab                 ;call macro
;
        org     p:start
;
        movep   x:txrx,x0           ;read pcm byte in x
        move    #>shift,y0          ;prepare 16 bit asr
        mpy     x0,y0,a   #>mask,x1 ;shift 16 bit in a
        and     x1,a      #>inv,y1  ;mask 8 lsb in a1
        eor     y1,a                ;invert even bit
        clr     a         a1,r3     ;set the table ptr
        nop                         ;wait one cycle
        move    x:(r3),a            ;a=decoded value
        end
```

```
        page    75,72,0,10
        opt     nomd,nomex,rc,mu
;
;ALGLN11.ASM
;alglin1.asm modified to work with half-size table.
;A-law to linear conversion using a table of data
;located in X ROM starting at location $180.
;Only positive values are included.
;
;    the 8 bit log data to be decoded
;    is in the ssi txrx register highbyte (bit23-bit15)
;
;       r3 points to the 13 bit linear data
; timing in cycles: 18 (-> 1.76 micro s)
; required memory space: - 128 data memory word
;                        -  13 program memory words
;REMARK: if R3:N3:M3 is only used in this routine,
;the instruction "move   #tab1,m3" can be put in some
;setup routine.
;
start   equ     $40  ;program starting address
tab     equ     $180 ;table starting address
tab1    equ     $7f  ;modulo value
shift   equ     $80  ;used in 16 bit asr & sign test
mask    equ     $7f  ;pcm byte mask
txrx    equ     $ffef     ;SSI rx register address
;
        org     p:start
;
        movep   x:txrx,x0       ;read pcm byte in x0
        move    #>shift,y0      ;prepare 16 bit asr
        mpy     x0,y0,a   #>mask,x1 ;16 bit asr in a
        and     x1,a      #tab,r3   ;mask 8 lsb in a1
        move    a1,n3               ;set the index
        move              #tab1,m3  ;r3 in modulo
        cmpm    y0,a                ;set ptr and cmp sgn
        move    x:(r3+n3),a     ;decoded pos. value in a
        jge     <endp           ;test pcm byte sign
        neg     a               ;negate if flag N set
endp
        end
```

ALGLIN2.ASM

This routine , which flowchart is in Fig. 10, decodes the PCM byte into linear data using the relation:

$$Lin = (2*Q + bias) * 2^{S-1}$$

where:

-Lin = 12 bit linear data (without sign bit)
-bias = 1 for S = 0
bias = 33hex for S = 1,7
-S = S2S1S0
-Q = Q3Q2Q1Q0

(see tables pp 5-9)

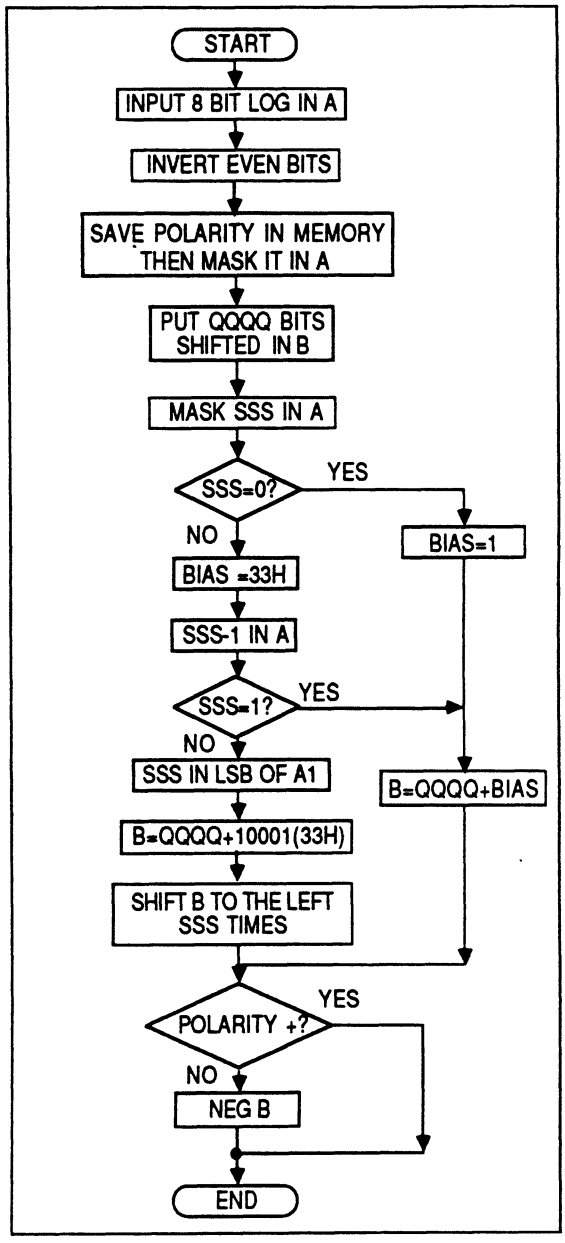

Fig. 10 . ALGLIN2.ASM flowchart

```
     page      75,72,0,10
;ALGLIN2.ASM
;  A-law to linear conversion algorithm
; -assumes the log 8 bit data to be decoded
;  present in the high byte of the ssi
;  txrx register(bit23-bit15)
; -result is given in the B accumulator on 13 bit
;worst case timing in cycles:   34 (-> 3.32 micro s.)
;required memory space:- 1 data memory word
;                       -28 program memory words
;WARNING: The memory location pointed by r0 will be used
start     equ       $40        ;program starting address
shift     equ       $08        ;used in  4 bit asr
shft1     equ       $000008    ;used in 20 bit asr
ofset     equ       $10         ;for test S2S1S0=1
bias      equ       $010800    ;;33 hex shifted msb=bit 11
one       equ       $000800    ;; 1     shifted msb=bit 11
maskb     equ       $00f000    ;;mask segment in b
mpq       equ       $70         ;;mask for q3q2q1q0
mp        equ       $7f         ;;mask for sign
inv       equ       $55         ;;invert bit 2,4,6,8
;
          org       p:start
          clr       a                  #inv,y1;clear a2:a1:a
          movep     x:txrx,a1         ;pcm byte in a
;    invert bits 2,4,6,8
          eor       y1,a       #<mp,y0          ;invert bit
                                                ;2,4,6,8
;    save polarity and mask it
          and       y0,a       a1,x:(r0)        ;a1=0sq
;    strip s in a and q in b
          move      a1,x0                  ;prepare a 4 bit
          move      #<shift,y1             ;right shift in b
          mpy       x0,y1,b    #maskb,y0   ;4 bit asr in b
          and       y0,b       #<mpq,y1    ;b=q3q2q1q0 shifted
          and       y1,a       #<ofset,x1       ;a=segment
          jne       <again                 ;jmp if not seg. 0
;    segment 0
          move      #one,y0                ;in seg. 0 bias=1
          jmp       <again1                ;skip seg. 1 test
again     sub       x1,a       #bias,y0    ;bias=10001h,test
          jgt       <shfta                 ;jmp if not seg. 1
;    segment 1
again1    add       y0,b                   ;add bias and goto
          jmp       <sgna                  ;sign without shift
;    other segments
shfta     move      a,x0       #>shft1,y1
          mpy       x0,y1,a         ;20 bit left shift
          add       y0,b            ;add bias in b
          rep       a1              ;seg. num.=shift counter
          lsl       b               ;shift b+bias
;     add polarity
sgna      jset #23,x:(r0),pos
          neg       b          ;negate result if negative
pos       nop
          end
```

2-b)A-law encoding routines

ALINLG1.ASM

Using the DSP56000/1 hardware Do loop instruction, this routine will have the segment number S2S1S0 successively in the loop counter LC and in the A accumulator extension byte A2. The flowchart is given in Fig. 11.

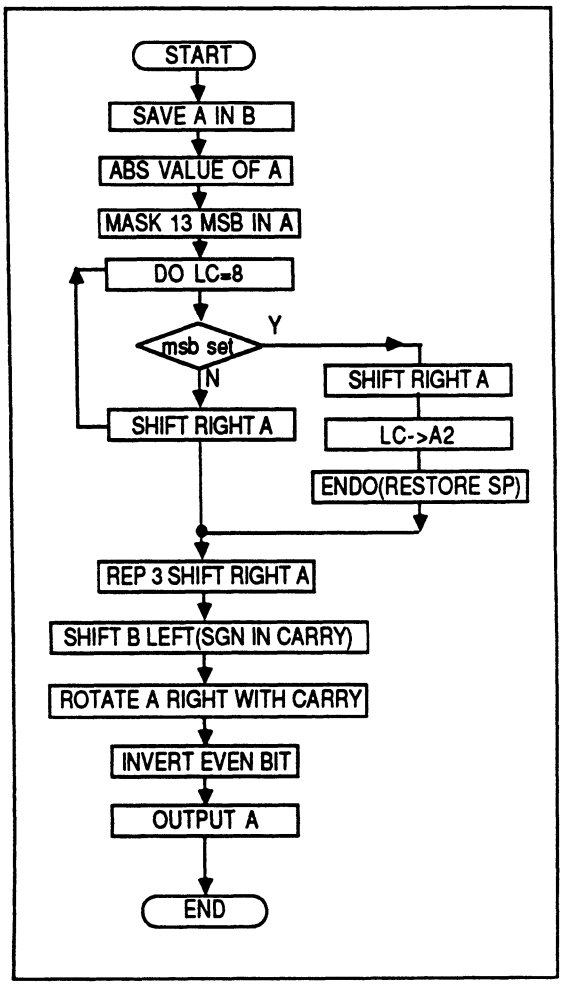

Fig. 11. ALINLG1.ASM flowchart

```
          opt       rc,mu
          page      75,72,0,10
;ALINLG1.ASM
;linear to logarithmic a-law conversion
;
;    the 13 bit linear input is in accumulator a
;    the 8  bit a-law is output to the ssi tx register
;worst case timing in cycles:   44 (-> 4.29 micro s.)
;required memory space:- 0 data memory word
;                        -18 program memory words
;
start     equ       $60         ;program starting address
txrx      equ       $ffef       ;SSI tx register address
msk13     equ       $fff800     ;truncate on the 13 msb
inv       equ       $d5         ;invert even bits + sign
;
          org       p:start
;
          tfr       a,b         #msk13,x0 ;b will store sign
          abs       a
          and       x0,a        #<inv,y1  ;only 13 msb remain
posit     do        #8,endlop             ;shift left until
          jes       <quit                 ;1 in a1 bit 22
          asl       a
endlop
          jmp       <seg0
quit      lsl       a           ;Q3Q2Q1Q0 in a1 4 msb
          move      lc,a2       ;lc= segment number in a2
          enddo                 ;restore in stack
seg0      rep       #3
          asr       a           ;S3S2S1Q3Q2Q1Q0 in a1 7 msb
;
          lsl       b           ;carry=sign of b
          ror       a           ;carry (sign)in a1 msb
          eor       y1,a        ;even bit + sign inversion
          movep     a1,x:txrx   ;output to SSI tx register
          end
```

ALINLG2.ASM

Faster than ALINLG2.ASM, this routine requests an 8 word data memory table to insert the segment number. The DSP56000/1 NORM instruction is used to correctly calculate the value to be pointed in the table, as shown on the flow chart (Fig. 12).

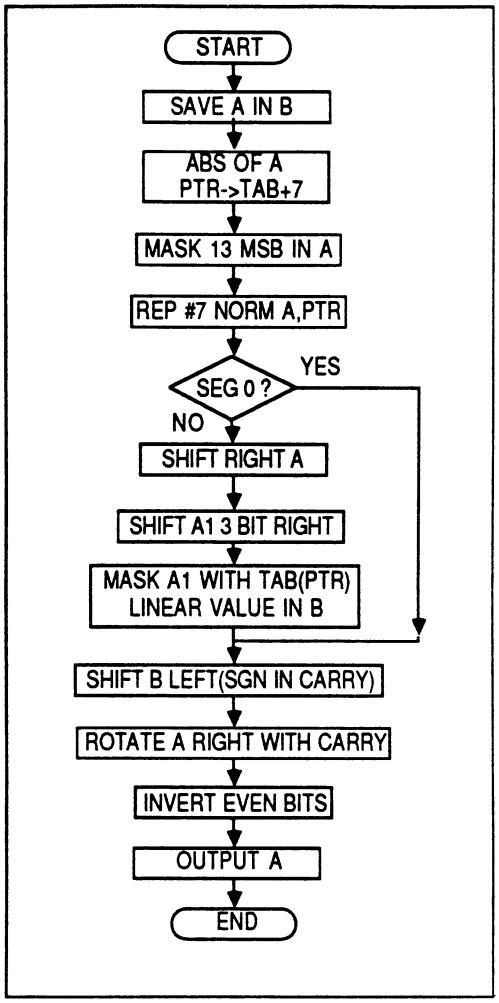

Fig. 12 . ALINLG2.ASM flowchart

```
            opt       rc,mu
            page      75,72,0,10
;ALINLG2.ASM
;linear to logarithmic a-law conversion
;    the 13 bit linear input is in accumulator a
;    the 8  bit a-law is output to the ssi tx register
;worst case timing in cycles:   26 (-> 2.54 micro s.)
;required memory space:- 8 data memory word
;                       -16 program memory words
;
start       equ       $60         ;program starting address
tab         equ       0           ;tab starting address
txrx        equ       $ffef       ;SSI tx register address
shft        equ       $20         ;used for 2 bit asr
msk13       equ       $fff800     ;truncate on 13 msb
inv         equ       $d5         ;invert even bits
;
            org       x:tab       ;this table contains masks
            dc        $1e0000     ;which operate on the 7 msb
            dc        $3e0000     ;of a1 which are supposed to
            dc        $5e0000     ;contain s2s1s0q3q2q1q0
            dc        $7e0000
            dc        $9e0000
            dc        $be0000
            dc        $de0000
            dc        $fe0000
;
            org       p:start
;
            tfr       a,b         #msk13,x0 ;b stores sign
            abs       a           #tab+7,r0 ;prepare ptr
            and       x0,a        #<inv,y1  ;mask the 11 lsb
;
            rep       #7          ;asl a1 until it contains
            norm      r0,a        ;01q3q2q1q0;each time r0-1
            jnn       <seg0       ;asl once more if s2s1s0=0
            lsl       a           ;a1=1q3q2q1q0
seg0        move      #<shft,y0
            move      a1,x0                     ;prepare 2 bit asr
            mpy       x0,y0,a     x:(r0),x1 ;x1=tab(r0)
            and       x1,a                      ;a1=s2s1s0q3q2q1q0

sgn         lsl       b                         ;carry = p
            ror       a                         ;a1=ps2s1s0q3q2q1q0
            eor       y1,a                      ;invert even bit
            movep     a1,x:txrx                 ;output to ssi
            end
```

3) MU-LAW CONVERSION ROUTINES

For each program, a flowchart given before the source listing will help the programer to understand the routines. The table MLOGLIN used by MLGLIN1 will be found in appendix C. Table MLGLNXP used in MLGLN11.ASM is exactly built with the last 128 elements of the former table and has therefore not been given as appendix. The fortran routine used to generate MLOGLIN.ASM is listed in appendix E. This routine can be modified in order to generate the table used by MLGLN11.ASM by changing two instructions:

-delete in WRITE 999 the macro statement and change "org x:tab" by "org x:$100",

-change "DO 20 I=1,256" by "DO 20 I=129,256"

The following table gives an overview of each techniques performance and requirements (time information is given for Fosc=20.5Mhz):

Function	Routine	Memory Size Prog	Data	Cycles	Time (micro s)
Expand	MLGLIN1	9	256	10	0.976
	MLGLN11	13	128	16	1.56
	MLGLIN2	22	1	33	3.22
Compress	MLINLG1	24	0	49	4.78
	MLINLG2	18	9	27	2.63

Table 11. Mu-Law conversion: Memory Space and Time Requirements

REMARK

According to CCITT recommendations [5] , when using the Mu-law in networks where suppression of the all 0 character signal is required, the byte 00 hex should be coded 02 hex (negative values between decision values number 127 and 128 will be coded 125).
This has not been done in the two companding routines given in this note and should be introduced by the programmer if necessary.

3-a)Mu-law decoding routines.

MLGLIN1.ASM & MLGLN11.ASM

The principle of these two routines is very simple. The PCM byte to be decoded is used to address a table in which the corresponding linear value has been stored.

MLGLIN1.ASM, which flowchart is in Fig. 13, uses a 256 element table.

MLGLN11.ASM uses the 128 element table (positive values only) located in DSP560001 X data ROM at starting address X:$100.

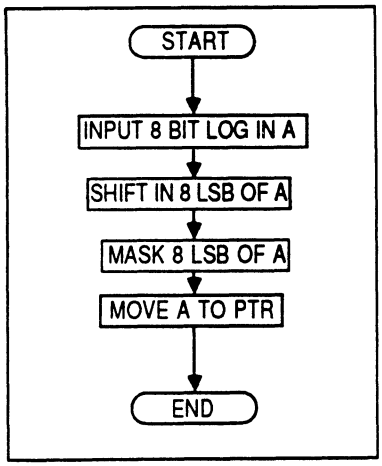

Fig.13 MLGLIN1.ASM flowchart

```
        opt  nomd,nomc,rc,mu
        page 75,72,0,10
;MLGLIN1.ASM
; mu-law to linear conversion using a table of data
;
; -8 bit data to be decoded is read
;  in the ssi txrx register highbyte (bit23-bit15)
; -r3 points  to the 13 bit decoded data
;
; timing in cycles:  10 (-> 976 ns)
; required memory space:256 data memory words
;                       9 program memory words
;
start     equ       $40          ;program starting address
tab       equ       0            ;table starting address
shift     equ       $80          ;16 bit asr
mask      equ       $ff          ;pcm byte mask
txrx      equ       $ffef        ;SSI rx register address
;
          maclib    '\dsp56000.mac\'    ;MS-DOS macro dir.
;
          mloglin   tab                 ;call macro
;
          org       p:start
;
          movep     x:txrx,x0           ;x0=pcm byte
          move      #>shift,y0          ;prepare 16 bit asr
          mpy       x0,y0,a   #>mask,x1 ;16 bit asr in a
          and       x1,a                ;mask 8 lsb in a1
          clr       a         a1,r3     ;ptr=pcm byte
          nop                           ;wait one cycle
          move      x:(r3),a            ;a=decoded value
          end
```

```
            opt       nomd,nomc,rc,mu
            page      75,72,0,10
; MLGLN11.ASM
; (mlglin1.asm modified to work with half table size)
; mu-law to linear conversion using a table of data
; located in X ROM starting at location HEX 100. Only the
; the positive values are included
; -8 bit data to be decoded is read
;  in the ssi txrx register highbyte (bit23-bit15)
; -r3 points to the 13 bit decoded data
;
;REMARK: if R3:N3:M3 is only used in this routine,
;the instruction "move   #tab1,m3" can be put in some
;initialization routine

start       equ       $40         ;program starting address
mask        equ       $7f         ;pcm byte mask
shift       equ       $80         ;16 bit asr & sign test
tab         equ       $100        ;table starting address
tab1        equ       $7f         ;modulo value
txrx        equ       $ffef       ;SSI rx register address
;
            org       p:start
;
            movep     x:txrx,x0             ;read pcm byte in x
            move      #>shift,y0            ;prepare 16 bit asr
            mpy       x0,y0,a   #>mask,x1 ;16 bit asr in a
            and       x1,a      #tab,r3   ;mask 8 lsb in a1
            move      a1,n3                 ;index=pcm byte
            move      #tab1,m3              ;modulo=$7f
            cmpm      y0,a                  ;set ptr and cmp sg
            move      x:(r3+n3),a     ;decoded pos. value in a
            jge       <endp           ;test pcm byte sgn
            neg       a               ;negate if N flag set
endp
            end
```

MLGLIN2.ASM

This routine, which flowchart is in Fig. 14, decodes the PCM byte into linear data using the relation:

$$Lin = (2*Q + bias)*2^{S} - bias$$

where:

- -Lin = 13 bit linear data (without sign bit)
- -bias = 33hex. for S = [0,7]
- -S = S2S1S0
- -Q = Q3Q2Q1Q0

(see tables pp 11-14)

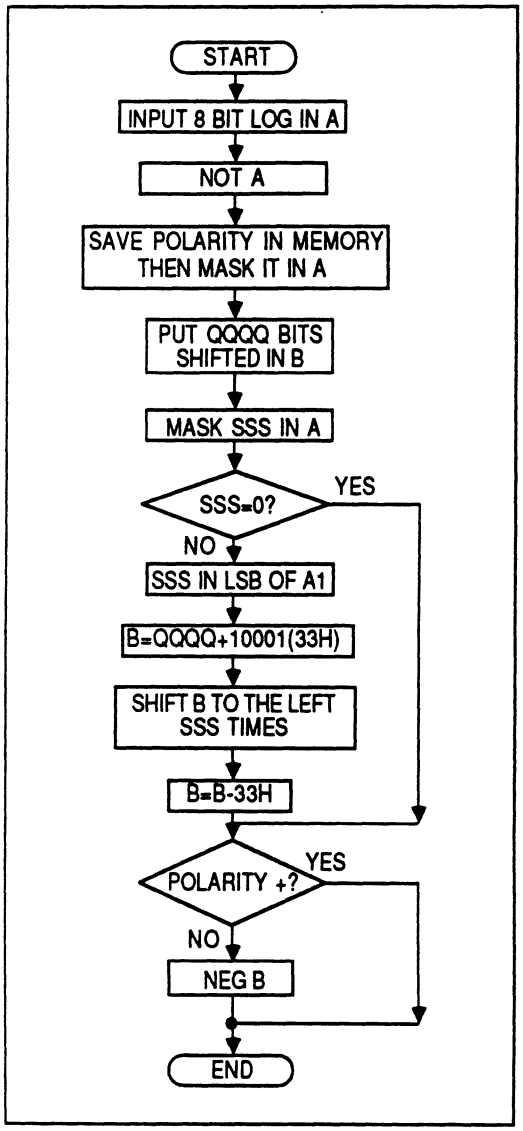

Fig. 14 . MLGLIN2.ASM flowchart

```
          page      75,72,0,10
;MLGLIN2.ASM
; -the 8 bit data to be decoded is read
;  in the highbyte (bit23-bit15)of the txrx ssi register
; -the result is given in accumulator b
;worst case timing in cycles: 33 (-> 3.22 micro s)
;required memory space:- 1 data memory word
;                       -22 program memory words
; ps:this routine uses a memory location pointed to by
;    r0 to store the polarity.this polarity can be saved
;    without using r0 in a direct addressing mode like
;    x:tab+8 if used with mlinlg2.asm
txrx      equ       $ffef     ;SSI tx register address
start     equ       $40       ;program starting address
shift     equ       $04       ;used for asr
shft1     equ       $000008   ;used for 20 bit asr
bias      equ       $008400   ;;33 shifted
maskb     equ       $007800   ;;mask s2s1s0 in b
mp        equ       $7f       ;;mask for p (sign)
mpq       equ       $70       ;;mask for p and q3q2q1q0
;
          org       p:start
;    invert input
          movep     x:txrx,a1             ;a1= pcm byte
          not       a         #<mp,y0     ;invert bit
;    save polarity and mask it
          and       y0,a      a1,x:(r0)   ;mask sign
;    strip s in a and q in b
          move      a1,x0               ;prepare a 5 bit
          move      #<shift,y1          ;asr in b
          mpy       x0,y1,b   #maskb,y0   ;b=q3q2q1q0 5 bit
          and       y0,b      #<mpq,y1    ;shifted and masked
          and       y1,a      #bias,y0    ;a=segment
          jeq       <sgn                  ;jmp if seg. 0
;    shift sum and remove bias
          move      a1,x0               ;a1 in x0
          move      #>shft1,y1          ;prepare a 20 bit asr
          mpy       x0,y1,a             ;seg. num. in a1 lsbyte
          add       y0,b                ;add bias to b
          rep       a1                  ;seg. num. = shift count
          lsl       b                   ;shift b+bias
          sub       y0,b                ;remove the bias
;    add polarity              b=ps2s1s0q3q2q1q0
sgn       jclr      #23,x:(r0),pos ;negate b
          neg       b                   ;if negative
pos       end
```

3-b)Mu-law encoding routines

MLINLG1.ASM

Using the DSP56000/1 hardware Do loop instruction, this routine will have the segment number S2S1S0 successively in the loop counter LC and in the A accumulator extension byte A2. The flowchart is given Fig. 15.

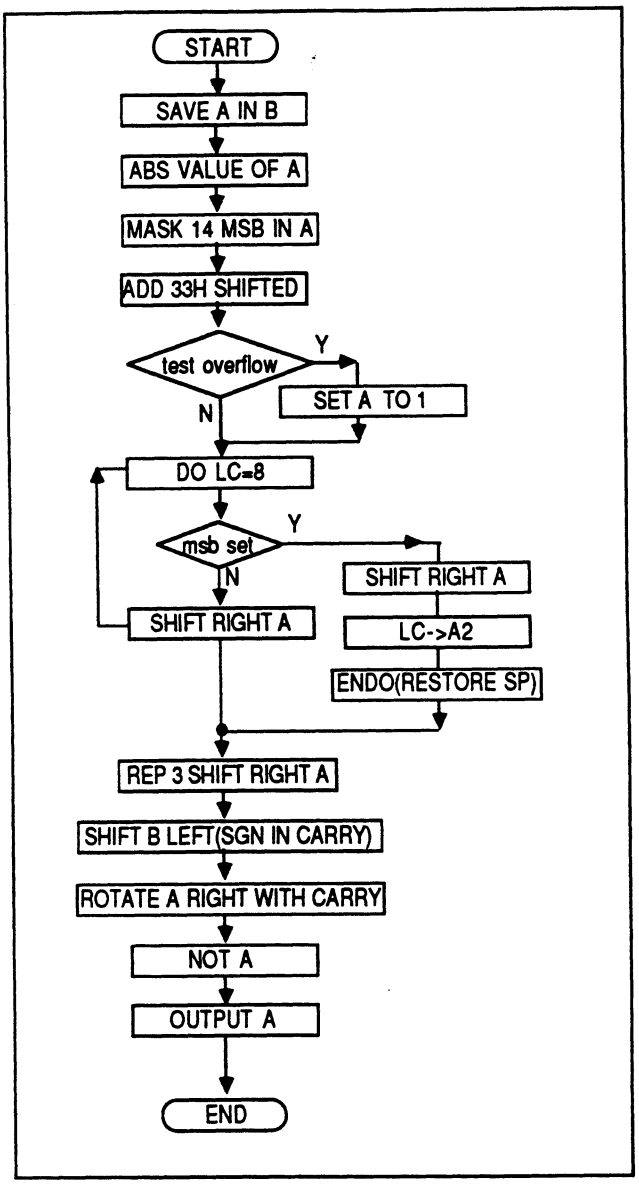

Fig. 15. MLINLG1.ASM flowchart

```
          opt       rc,mu
          page      75,72,0,10
;MLINLG1.ASM
;linear to logarithmic mu-law conversion
;    the 14 bit linear data is in accumulator a
;    the 8 bit  data is output to ssi tx register
;worst case timing in cycles: 49 (-> 4.78 micro s)
;required memory space:- 0 data memory word
;                        -24 program memory words
;
start     equ       $60
txrx      equ       $ffef     ;SSI tx register address
msk14     equ       $fffc00   ;mask 10 lsb bit
bias      equ       $008400   ;33 shifted (lsb=bit 10)
;
          org       p:start
;
          tfr       a,b       #msk14,x0 ;b store sign
          abs       a                   ;absolute value
          and       x0,a      #bias,y0  ;14 b truncation
          add       y0,a                ;add bias
          jec       <novflw                 ;if overflow
          rnd       b         #$7fffff,a    ;saturation
;
novflw    do        #8,endloop      ;shift left until
          jes       <quit           ;1 in a1 bit 22
          asl       a               ;lc will be the segment
endloop
          lsl       a               ;a1=1q3q2q1q0
          jmp       <seg0
;
quit      lsl       a               ;a1=q3q2q1q0
          move      lc,a2           ;a2=seg. number
          enddo                     ;restore stack
;
seg0      rep       #3
          asr       a         ;a1=s2s1s0q3q2q1q0
          lsl       b         ;carry=p
          ror       a         ;a1=ps2s1s0q3q2q1q0
          not       a         ;bit inversion before
          movep     a1,x:txrx ;output in SSI tx
          end
```

Faster than MLINLG1.ASM, this routine requests an 8 word data memory table to insert the segment number. DSP56000/1 NORM instruction is used to correctly calculate the value to be pointed in the table, as shown on the flowchart (Fig. 16).

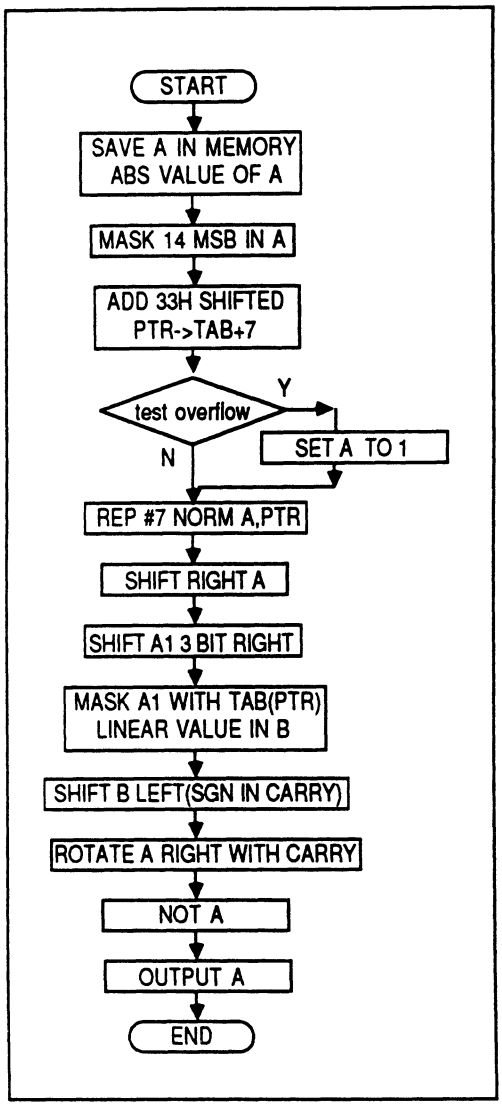

Fig. 16 . MLINLG2.ASM flowchart

```
;MLINLG2.ASM
;linear to logarithmic mu-law conversion
;    the 14 bit linear data is in accumulator a
;    the 8 bit  data is output to ssi tx register
;worst case timing in cycles: 27 (-> 2.63 micro s)
;required memory space:- 9 data memory word
;                       -18 program memory words

          page      75,72,0,10
          opt       rc,mu
txrx      equ       $ffef       ;SSI tx register address
shft      equ       $20         ;used for 2 bit asr
msk14     equ       $fffc00     ;truncate on 14 msb
tab       equ       $0          ;table starting address
start     equ       $60         ;program starting address
bias      equ       $008400     ;33h shifted with lsb= bit 10

          org       x:tab       ;this table contains masks
          dc        $1e0000     ;which operate on the 7 msb
          dc        $3e0000     ;of a1 which are supposed to
          dc        $5e0000     ;contain s2s1s0q3q2q1q0.
          dc        $7e0000
          dc        $9e0000
          dc        $be0000
          dc        $de0000
          dc        $fe0000
;
          org       p:start
          move      #msk14,x0
          abs       a           a,x:tab+8 ;x:tab+8 store sign
          and       x0,a        #bias,y0  ;mask abs(a) 10lsb
          add       y0,a        #tab+7,r0 ;init ptr&add bias
          jec       <novflw               ;if overflow
          clr       b           x:tab+3,a ;a1=7e (all one)
novflw    rep       #7          ;asl a until it contains
          norm      r0,a        ;01q3q2q1q0/each time r0-1
          asl       a           #<shft,y0 ;a1=1q3q2q1q0
          move                  a1,x0     ;for 2 bit asr
          mpy       x0,y0,a     x:(r0),x1 ;x1=tab(r0)
          and       x1,a        x:tab+8,b ;a=s2s1s0q3q2q1q0
          lsl       b                     ;carry= p(sign)
          ror       a                     ;a=ps2s1s0q3q2q1q0
          not       a                     ;bit invert before
          movep     a1,x:txrx             ;output to SSI tx
          end
```

APPENDIX A

MACRO ALOGLIN (file ALOGLIN.ASM)

```
;this macro contains the A-law
decoding table
;it should be called with the
starting address tab

aloglin macro  tab

        org    x:tab

A_0     DC     $FFF800 ;   -1
A_1     DC     $FFE800 ;   -3
A_2     DC     $FFD800 ;   -5
A_3     DC     $FFC800 ;   -7
A_4     DC     $FFB800 ;   -9
A_5     DC     $FFA800 ;  -11
A_6     DC     $FF9800 ;  -13
A_7     DC     $FF8800 ;  -15
A_8     DC     $FF7800 ;  -17
A_9     DC     $FF6800 ;  -19
A_A     DC     $FF5800 ;  -21
A_B     DC     $FF4800 ;  -23
A_C     DC     $FF3800 ;  -25
A_D     DC     $FF2800 ;  -27
A_E     DC     $FF1800 ;  -29
A_F     DC     $FF0800 ;  -31
A_10    DC     $FEF800 ;  -33
A_11    DC     $FEE800 ;  -35
A_12    DC     $FED800 ;  -37
A_13    DC     $FEC800 ;  -39
A_14    DC     $FEB800 ;  -41
A_15    DC     $FEA800 ;  -43
A_16    DC     $FE9800 ;  -45
A_17    DC     $FE8800 ;  -47
A_18    DC     $FE7800 ;  -49
A_19    DC     $FE6800 ;  -51
A_1A    DC     $FE5800 ;  -53
A_1B    DC     $FE4800 ;  -55
A_1C    DC     $FE3800 ;  -57
A_1D    DC     $FE2800 ;  -59
A_1E    DC     $FE1800 ;  -61
A_1F    DC     $FE0800 ;  -63
A_20    DC     $FDF000 ;  -66
A_21    DC     $FDD000 ;  -70
A_22    DC     $FDB000 ;  -74
A_23    DC     $FD9000 ;  -78
A_24    DC     $FD7000 ;  -82
A_25    DC     $FD5000 ;  -86
A_26    DC     $FD3000 ;  -90
A_27    DC     $FD1000 ;  -94
A_28    DC     $FCF000 ;  -98
A_29    DC     $FCD000 ; -102
A_2A    DC     $FCB000 ; -106
A_2B    DC     $FC9000 ; -110
A_2C    DC     $FC7000 ; -114
A_2D    DC     $FC5000 ; -118
A_2E    DC     $FC3000 ; -122
A_2F    DC     $FC1000 ; -126
A_30    DC     $FBE000 ; -132
A_31    DC     $FBA000 ; -140
A_32    DC     $FB6000 ; -148
A_33    DC     $FB2000 ; -156
A_34    DC     $FAE000 ; -164
A_35    DC     $FAA000 ; -172
A_36    DC     $FA6000 ; -180
A_37    DC     $FA2000 ; -188
A_38    DC     $F9E000 ; -196
A_39    DC     $F9A000 ; -204
A_3A    DC     $F96000 ; -212
A_3B    DC     $F92000 ; -220
A_3C    DC     $F8E000 ; -228
A_3D    DC     $F8A000 ; -236
A_3E    DC     $F86000 ; -244
A_3F    DC     $F82000 ; -252
A_40    DC     $F7C000 ; -264
A_41    DC     $F74000 ; -280
A_42    DC     $F6C000 ; -296
A_43    DC     $F64000 ; -312
A_44    DC     $F5C000 ; -328
A_45    DC     $F54000 ; -344
A_46    DC     $F4C000 ; -360
A_47    DC     $F44000 ; -376
A_48    DC     $F3C000 ; -392
A_49    DC     $F34000 ; -408
A_4A    DC     $F2C000 ; -424
A_4B    DC     $F24000 ; -440
A_4C    DC     $F1C000 ; -456
A_4D    DC     $F14000 ; -472
A_4E    DC     $F0C000 ; -488
A_4F    DC     $F04000 ; -504
A_50    DC     $EF8000 ; -528
A_51    DC     $EE8000 ; -560
A_52    DC     $ED8000 ; -592
A_53    DC     $EC8000 ; -624
A_54    DC     $EB8000 ; -656
A_55    DC     $EA8000 ; -688
A_56    DC     $E98000 ; -720
A_57    DC     $E88000 ; -752
A_58    DC     $E78000 ; -784
```

```
A_59    DC    $E68000 ; -816     A_93    DC    $013800 ;   39
A_5A    DC    $E58000 ; -848     A_94    DC    $014800 ;   41
A_5B    DC    $E48000 ; -880     A_95    DC    $015800 ;   43
A_5C    DC    $E38000 ; -912     A_96    DC    $016800 ;   45
A_5D    DC    $E28000 ; -944     A_97    DC    $017800 ;   47
A_5E    DC    $E18000 ; -976     A_98    DC    $018800 ;   49
A_5F    DC    $E08000 ;-1008     A_99    DC    $019800 ;   51
A_60    DC    $DF0000 ;-1056     A_9A    DC    $01A800 ;   53
A_61    DC    $DD0000 ;-1120     A_9B    DC    $01B800 ;   55
A_62    DC    $DB0000 ;-1184     A_9C    DC    $01C800 ;   57
A_63    DC    $D90000 ;-1248     A_9D    DC    $01D800 ;   59
A_64    DC    $D70000 ;-1312     A_9E    DC    $01E800 ;   61
A_65    DC    $D50000 ;-1376     A_9F    DC    $01F800 ;   63
A_66    DC    $D30000 ;-1440     A_A0    DC    $021000 ;   66
A_67    DC    $D10000 ;-1504     A_A1    DC    $023000 ;   70
A_68    DC    $CF0000 ;-1568     A_A2    DC    $025000 ;   74
A_69    DC    $CD0000 ;-1632     A_A3    DC    $027000 ;   78
A_6A    DC    $CB0000 ;-1696     A_A4    DC    $029000 ;   82
A_6B    DC    $C90000 ;-1760     A_A5    DC    $02B000 ;   86
A_6C    DC    $C70000 ;-1824     A_A6    DC    $02D000 ;   90
A_6D    DC    $C50000 ;-1888     A_A7    DC    $02F000 ;   94
A_6E    DC    $C30000 ;-1952     A_A8    DC    $031000 ;   98
A_6F    DC    $C10000 ;-2016     A_A9    DC    $033000 ;  102
A_70    DC    $BE0000 ;-2112     A_AA    DC    $035000 ;  106
A_71    DC    $BA0000 ;-2240     A_AB    DC    $037000 ;  110
A_72    DC    $B60000 ;-2368     A_AC    DC    $039000 ;  114
A_73    DC    $B20000 ;-2496     A_AD    DC    $03B000 ;  118
A_74    DC    $AE0000 ;-2624     A_AE    DC    $03D000 ;  122
A_75    DC    $AA0000 ;-2752     A_AF    DC    $03F000 ;  126
A_76    DC    $A60000 ;-2880     A_B0    DC    $042000 ;  132
A_77    DC    $A20000 ;-3008     A_B1    DC    $046000 ;  140
A_78    DC    $9E0000 ;-3136     A_B2    DC    $04A000 ;  148
A_79    DC    $9A0000 ;-3264     A_B3    DC    $04E000 ;  156
A_7A    DC    $960000 ;-3392     A_B4    DC    $052000 ;  164
A_7B    DC    $920000 ;-3520     A_B5    DC    $056000 ;  172
A_7C    DC    $8E0000 ;-3648     A_B6    DC    $05A000 ;  180
A_7D    DC    $8A0000 ;-3776     A_B7    DC    $05E000 ;  188
A_7E    DC    $860000 ;-3904     A_B8    DC    $062000 ;  196
A_7F    DC    $820000 ;-4032     A_B9    DC    $066000 ;  204
A_80    DC    $000800 ;    1     A_BA    DC    $06A000 ;  212
A_81    DC    $001800 ;    3     A_BB    DC    $06E000 ;  220
A_82    DC    $002800 ;    5     A_BC    DC    $072000 ;  228
A_83    DC    $003800 ;    7     A_BD    DC    $076000 ;  236
A_84    DC    $004800 ;    9     A_BE    DC    $07A000 ;  244
A_85    DC    $005800 ;   11     A_BF    DC    $07E000 ;  252
A_86    DC    $006800 ;   13     A_C0    DC    $084000 ;  264
A_87    DC    $007800 ;   15     A_C1    DC    $08C000 ;  280
A_88    DC    $008800 ;   17     A_C2    DC    $094000 ;  296
A_89    DC    $009800 ;   19     A_C3    DC    $09C000 ;  312
A_8A    DC    $00A800 ;   21     A_C4    DC    $0A4000 ;  328
A_8B    DC    $00B800 ;   23     A_C5    DC    $0AC000 ;  344
A_8C    DC    $00C800 ;   25     A_C6    DC    $0B4000 ;  360
A_8D    DC    $00D800 ;   27     A_C7    DC    $0BC000 ;  376
A_8E    DC    $00E800 ;   29     A_C8    DC    $0C4000 ;  392
A_8F    DC    $00F800 ;   31     A_C9    DC    $0CC000 ;  408
A_90    DC    $010800 ;   33     A_CA    DC    $0D4000 ;  424
A_91    DC    $011800 ;   35     A_CB    DC    $0DC000 ;  440
A_92    DC    $012800 ;   37     A_CC    DC    $0E4000 ;  456
```

```
A_CD    DC    $0EC000 ;  472    A_E7    DC    $2F0000 ; 1504
A_CE    DC    $0F4000 ;  488    A_E8    DC    $310000 ; 1568
A_CF    DC    $0FC000 ;  504    A_E9    DC    $330000 ; 1632
A_D0    DC    $108000 ;  528    A_EA    DC    $350000 ; 1696
A_D1    DC    $118000 ;  560    A_EB    DC    $370000 ; 1760
A_D2    DC    $128000 ;  592    A_EC    DC    $390000 ; 1824
A_D3    DC    $138000 ;  624    A_ED    DC    $3B0000 ; 1888
A_D4    DC    $148000 ;  656    A_EE    DC    $3D0000 ; 1952
A_D5    DC    $158000 ;  688    A_EF    DC    $3F0000 ; 2016
A_D6    DC    $168000 ;  720    A_F0    DC    $420000 ; 2112
A_D7    DC    $178000 ;  752    A_F1    DC    $460000 ; 2240
A_D8    DC    $188000 ;  784    A_F2    DC    $4A0000 ; 2368
A_D9    DC    $198000 ;  816    A_F3    DC    $4E0000 ; 2496
A_DA    DC    $1A8000 ;  848    A_F4    DC    $520000 ; 2624
A_DB    DC    $1B8000 ;  880    A_F5    DC    $560000 ; 2752
A_DC    DC    $1C8000 ;  912    A_F6    DC    $5A0000 ; 2880
A_DD    DC    $1D8000 ;  944    A_F7    DC    $5E0000 ; 3008
A_DE    DC    $1E8000 ;  976    A_F8    DC    $620000 ; 3136
A_DF    DC    $1F8000 ; 1008    A_F9    DC    $660000 ; 3264
A_E0    DC    $210000 ; 1056    A_FA    DC    $6A0000 ; 3392
A_E1    DC    $230000 ; 1120    A_FB    DC    $6E0000 ; 3520
A_E2    DC    $250000 ; 1184    A_FC    DC    $720000 ; 3648
A_E3    DC    $270000 ; 1248    A_FD    DC    $760000 ; 3776
A_E4    DC    $290000 ; 1312    A_FE    DC    $7A0000 ; 3904
A_E5    DC    $2B0000 ; 1376    A_FF    DC    $7E0000 ; 4032
A_E6    DC    $2D0000 ; 1440    ;
                                        ENDM
                                ;
```

APPENDIX B

ALGLNXP.ASM

This program contains the table generated by ALGLNX.FOR and used by the routine ALGLN11.ASM.
The table elements are the decoder output values for the 128 positive segments ordered in the inverted even bit. This table is inserted into the internal DSP56000/1 X data memory , between the addresses $180-$1ff.

```
     opt rc
;alglnxp.asm
;this routine contains the half
A-law decoding table
;the table is in the even bit
order and decodes
;directly the positive segments
;it is located in the internal X
data memory
;with the starting address X:180
hex.

        org    X:$180

;
A_80    DC    $158000 ;  688
A_81    DC    $148000 ;  656
A_82    DC    $178000 ;  752
A_83    DC    $168000 ;  720
A_84    DC    $118000 ;  560
A_85    DC    $108000 ;  528
A_86    DC    $138000 ;  624
A_87    DC    $128000 ;  592
A_88    DC    $1D8000 ;  944
A_89    DC    $1C8000 ;  912
A_8A    DC    $1F8000 ; 1008
A_8B    DC    $1E8000 ;  976
A_8C    DC    $198000 ;  816
A_8D    DC    $188000 ;  784
A_8E    DC    $1B8000 ;  880
A_8F    DC    $1A8000 ;  848
A_90    DC    $0AC000 ;  344
A_91    DC    $0A4000 ;  328
A_92    DC    $0BC000 ;  376
A_93    DC    $0B4000 ;  360
A_94    DC    $08C000 ;  280
```

```
A_95    DC    $084000 ;  264    A_CB    DC    $01E800 ;   61
A_96    DC    $09C000 ;  312    A_CC    DC    $019800 ;   51
A_97    DC    $094000 ;  296    A_CD    DC    $018800 ;   49
A_98    DC    $0EC000 ;  472    A_CE    DC    $01B800 ;   55
A_99    DC    $0E4000 ;  456    A_CF    DC    $01A800 ;   53
A_9A    DC    $0FC000 ;  504    A_D0    DC    $005800 ;   11
A_9B    DC    $0F4000 ;  488    A_D1    DC    $004800 ;    9
A_9C    DC    $0CC000 ;  408    A_D2    DC    $007800 ;   15
A_9D    DC    $0C4000 ;  392    A_D3    DC    $006800 ;   13
A_9E    DC    $0DC000 ;  440    A_D4    DC    $001800 ;    3
A_9F    DC    $0D4000 ;  424    A_D5    DC    $000800 ;    1
A_A0    DC    $560000 ; 2752    A_D6    DC    $003800 ;    7
A_A1    DC    $520000 ; 2624    A_D7    DC    $002800 ;    5
A_A2    DC    $5E0000 ; 3008    A_D8    DC    $00D800 ;   27
A_A3    DC    $5A0000 ; 2880    A_D9    DC    $00C800 ;   25
A_A4    DC    $460000 ; 2240    A_DA    DC    $00F800 ;   31
A_A5    DC    $420000 ; 2112    A_DB    DC    $00E800 ;   29
A_A6    DC    $4E0000 ; 2496    A_DC    DC    $009800 ;   19
A_A7    DC    $4A0000 ; 2368    A_DD    DC    $008800 ;   17
A_A8    DC    $760000 ; 3776    A_DE    DC    $00B800 ;   23
A_A9    DC    $720000 ; 3648    A_DF    DC    $00A800 ;   21
A_AA    DC    $7E0000 ; 4032    A_E0    DC    $056000 ;  172
A_AB    DC    $7A0000 ; 3904    A_E1    DC    $052000 ;  164
A_AC    DC    $660000 ; 3264    A_E2    DC    $05E000 ;  188
A_AD    DC    $620000 ; 3136    A_E3    DC    $05A000 ;  180
A_AE    DC    $6E0000 ; 3520    A_E4    DC    $046000 ;  140
A_AF    DC    $6A0000 ; 3392    A_E5    DC    $042000 ;  132
A_B0    DC    $2B0000 ; 1376    A_E6    DC    $04E000 ;  156
A_B1    DC    $290000 ; 1312    A_E7    DC    $04A000 ;  148
A_B2    DC    $2F0000 ; 1504    A_E8    DC    $076000 ;  236
A_B3    DC    $2D0000 ; 1440    A_E9    DC    $072000 ;  228
A_B4    DC    $230000 ; 1120    A_EA    DC    $07E000 ;  252
A_B5    DC    $210000 ; 1056    A_EB    DC    $07A000 ;  244
A_B6    DC    $270000 ; 1248    A_EC    DC    $066000 ;  204
A_B7    DC    $250000 ; 1184    A_ED    DC    $062000 ;  196
A_B8    DC    $3B0000 ; 1888    A_EE    DC    $06E000 ;  220
A_B9    DC    $390000 ; 1824    A_EF    DC    $06A000 ;  212
A_BA    DC    $3F0000 ; 2016    A_F0    DC    $02B000 ;   86
A_BB    DC    $3D0000 ; 1952    A_F1    DC    $029000 ;   82
A_BC    DC    $330000 ; 1632    A_F2    DC    $02F000 ;   94
A_BD    DC    $310000 ; 1568    A_F3    DC    $02D000 ;   90
A_BE    DC    $370000 ; 1760    A_F4    DC    $023000 ;   70
A_BF    DC    $350000 ; 1696    A_F5    DC    $021000 ;   66
A_C0    DC    $015800 ;   43    A_F6    DC    $027000 ;   78
A_C1    DC    $014800 ;   41    A_F7    DC    $025000 ;   74
A_C2    DC    $017800 ;   47    A_F8    DC    $03B000 ;  118
A_C3    DC    $016800 ;   45    A_F9    DC    $039000 ;  114
A_C4    DC    $011800 ;   35    A_FA    DC    $03F000 ;  126
A_C5    DC    $010800 ;   33    A_FB    DC    $03D000 ;  122
A_C6    DC    $013800 ;   39    A_FC    DC    $033000 ;  102
A_C7    DC    $012800 ;   37    A_FD    DC    $031000 ;   98
A_C8    DC    $01D800 ;   59    A_FE    DC    $037000 ;  110
A_C9    DC    $01C800 ;   57    A_FF    DC    $035000 ;  106
A_CA    DC    $01F800 ;   63    ;

                                        END
                                ;
```

MLOGLIN MACRO (file MLOGLIN.ASM)

```
;this macro contains the mu-law
decoding table
;it should be called with the
starting address tab

mloglin macro  tab

        org    x:tab

;
U_0     DC     $828400 ;-8031
U_1     DC     $868400 ;-7775
U_2     DC     $8A8400 ;-7519
U_3     DC     $8E8400 ;-7263
U_4     DC     $928400 ;-7007
U_5     DC     $968400 ;-6751
U_6     DC     $9A8400 ;-6495
U_7     DC     $9E8400 ;-6239
U_8     DC     $A28400 ;-5983
U_9     DC     $A68400 ;-5727
U_A     DC     $AA8400 ;-5471
M_B     DC     $AE8400 ;-5215
U_C     DC     $B28400 ;-4959
U_D     DC     $B68400 ;-4703
U_E     DC     $BA8400 ;-4447
U_F     DC     $BE8400 ;-4191
U_10    DC     $C18400 ;-3999
U_11    DC     $C38400 ;-3871
U_12    DC     $C58400 ;-3743
U_13    DC     $C78400 ;-3615
U_14    DC     $C98400 ;-3487
U_15    DC     $CB8400 ;-3359
U_16    DC     $CD8400 ;-3231
U_17    DC     $CF8400 ;-3103
U_18    DC     $D18400 ;-2975
U_19    DC     $D38400 ;-2847
U_1A    DC     $D58400 ;-2719
U_1B    DC     $D78400 ;-2591
U_1C    DC     $D98400 ;-2463
U_1D    DC     $DB8400 ;-2335
U_1E    DC     $DD8400 ;-2207
U_1F    DC     $DF8400 ;-2079
U_20    DC     $E10400 ;-1983
U_21    DC     $E20400 ;-1919
U_22    DC     $E30400 ;-1855
U_23    DC     $E40400 ;-1791
U_24    DC     $E50400 ;-1727
U_25    DC     $E60400 ;-1663
U_26    DC     $E70400 ;-1599
U_27    DC     $E80400 ;-1535
U_28    DC     $E90400 ;-1471
U_29    DC     $EA0400 ;-1407
U_2A    DC     $EB0400 ;-1343
U_2B    DC     $EC0400 ;-1279
U_2C    DC     $ED0400 ;-1215
U_2D    DC     $EE0400 ;-1151
U_2E    DC     $EF0400 ;-1087
U_2F    DC     $F00400 ;-1023
U_30    DC     $F0C400 ; -975
U_31    DC     $F14400 ; -943
U_32    DC     $F1C400 ; -911
U_33    DC     $F24400 ; -879
U_34    DC     $F2C400 ; -847
U_35    DC     $F34400 ; -815
U_36    DC     $F3C400 ; -783
U_37    DC     $F44400 ; -751
U_38    DC     $F4C400 ; -719
U_39    DC     $F54400 ; -687
U_3A    DC     $F5C400 ; -655
U_3B    DC     $F64400 ; -623
U_3C    DC     $F6C400 ; -591
U_3D    DC     $F74400 ; -559
U_3E    DC     $F7C400 ; -527
U_3F    DC     $F84400 ; -495
U_40    DC     $F8A400 ; -471
U_41    DC     $F8E400 ; -455
U_42    DC     $F92400 ; -439
U_43    DC     $F96400 ; -423
U_44    DC     $F9A400 ; -407
U_45    DC     $F9E400 ; -391
U_46    DC     $FA2400 ; -375
U_47    DC     $FA6400 ; -359
U_48    DC     $FAA400 ; -343
U_49    DC     $FAE400 ; -327
U_4A    DC     $FB2400 ; -311
U_4B    DC     $FB6400 ; -295
U_4C    DC     $FBA400 ; -279
U_4D    DC     $FBE400 ; -263
U_4E    DC     $FC2400 ; -247
U_4F    DC     $FC6400 ; -231
U_50    DC     $FC9400 ; -219
U_51    DC     $FCB400 ; -211
U_52    DC     $FCD400 ; -203
U_53    DC     $FCF400 ; -195
```

```
U_54    DC    $FD1400 ; -187     U_8E    DC    $457C00 ; 4447
U_55    DC    $FD3400 ; -179     U_8F    DC    $417C00 ; 4191
U_56    DC    $FD5400 ; -171     U_90    DC    $3E7C00 ; 3999
U_57    DC    $FD7400 ; -163     U_91    DC    $3C7C00 ; 3871
U_58    DC    $FD9400 ; -155     U_92    DC    $3A7C00 ; 3743
U_59    DC    $FDB400 ; -147     U_93    DC    $387C00 ; 3615
U_5A    DC    $FDD400 ; -139     U_94    DC    $367C00 ; 3487
U_5B    DC    $FDF400 ; -131     U_95    DC    $347C00 ; 3359
U_5C    DC    $FE1400 ; -123     U_96    DC    $327C00 ; 3231
U_5D    DC    $FE3400 ; -115     U_97    DC    $307C00 ; 3103
U_5E    DC    $FE5400 ; -107     U_98    DC    $2E7C00 ; 2975
U_5F    DC    $FE7400 ;  -99     U_99    DC    $2C7C00 ; 2847
U_60    DC    $FE8C00 ;  -93     U_9A    DC    $2A7C00 ; 2719
U_61    DC    $FE9C00 ;  -89     U_9B    DC    $287C00 ; 2591
U_62    DC    $FEAC00 ;  -85     U_9C    DC    $267C00 ; 2463
U_63    DC    $FEBC00 ;  -81     U_9D    DC    $247C00 ; 2335
U_64    DC    $FECC00 ;  -77     U_9E    DC    $227C00 ; 2207
U_65    DC    $FEDC00 ;  -73     U_9F    DC    $207C00 ; 2079
U_66    DC    $FEEC00 ;  -69     U_A0    DC    $1EFC00 ; 1983
U_67    DC    $FEFC00 ;  -65     U_A1    DC    $1DFC00 ; 1919
U_68    DC    $FF0C00 ;  -61     U_A2    DC    $1CFC00 ; 1855
U_69    DC    $FF1C00 ;  -57     U_A3    DC    $1BFC00 ; 1791
U_6A    DC    $FF2C00 ;  -53     U_A4    DC    $1AFC00 ; 1727
U_6B    DC    $FF3C00 ;  -49     U_A5    DC    $19FC00 ; 1663
U_6C    DC    $FF4C00 ;  -45     U_A6    DC    $18FC00 ; 1599
U_6D    DC    $FF5C00 ;  -41     U_A7    DC    $17FC00 ; 1535
U_6E    DC    $FF6C00 ;  -37     U_A8    DC    $16FC00 ; 1471
U_6F    DC    $FF7C00 ;  -33     U_A9    DC    $15FC00 ; 1407
U_70    DC    $FF8800 ;  -30     U_AA    DC    $14FC00 ; 1343
U_71    DC    $FF9000 ;  -28     U_AB    DC    $13FC00 ; 1279
U_72    DC    $FF9800 ;  -26     U_AC    DC    $12FC00 ; 1215
U_73    DC    $FFA000 ;  -24     U_AD    DC    $11FC00 ; 1151
U_74    DC    $FFA800 ;  -22     U_AE    DC    $10FC00 ; 1087
U_75    DC    $FFB000 ;  -20     U_AF    DC    $0FFC00 ; 1023
U_76    DC    $FFB800 ;  -18     U_B0    DC    $0F3C00 ;  975
U_77    DC    $FFC000 ;  -16     U_B1    DC    $0EBC00 ;  943
U_78    DC    $FFC800 ;  -14     U_B2    DC    $0E3C00 ;  911
U_79    DC    $FFD000 ;  -12     U_B3    DC    $0DBC00 ;  879
U_7A    DC    $FFD800 ;  -10     U_B4    DC    $0D3C00 ;  847
U_7B    DC    $FFE000 ;   -8     U_B5    DC    $0CBC00 ;  815
U_7C    DC    $FFE800 ;   -6     U_B6    DC    $0C3C00 ;  783
U_7D    DC    $FFF000 ;   -4     U_B7    DC    $0BBC00 ;  751
U_7E    DC    $FFF800 ;   -2     U_B8    DC    $0B3C00 ;  719
U_7F    DC    $000000 ;    0     U_B9    DC    $0ABC00 ;  687
U_80    DC    $7D7C00 ; 8031     U_BA    DC    $0A3C00 ;  655
U_81    DC    $797C00 ; 7775     U_BB    DC    $09BC00 ;  623
U_82    DC    $757C00 ; 7519     U_BC    DC    $093C00 ;  591
U_83    DC    $717C00 ; 7263     U_BD    DC    $08BC00 ;  559
U_84    DC    $6D7C00 ; 7007     U_BE    DC    $083C00 ;  527
U_85    DC    $697C00 ; 6751     U_BF    DC    $07BC00 ;  495
U_86    DC    $657C00 ; 6495     U_C0    DC    $075C00 ;  471
U_87    DC    $617C00 ; 6239     U_C1    DC    $071C00 ;  455
U_88    DC    $5D7C00 ; 5983     U_C2    DC    $06DC00 ;  439
U_89    DC    $597C00 ; 5727     U_C3    DC    $069C00 ;  423
U_8A    DC    $557C00 ; 5471     U_C4    DC    $065C00 ;  407
U_8B    DC    $517C00 ; 5215     U_C5    DC    $061C00 ;  391
U_8C    DC    $4D7C00 ; 4959     U_C6    DC    $05DC00 ;  375
U_8D    DC    $497C00 ; 4703     U_C7    DC    $059C00 ;  359
```

```
U_C8    DC      $055C00 ;  343      U_E6    DC      $011400 ;   69
U_C9    DC      $051C00 ;  327      U_E7    DC      $010400 ;   65
U_CA    DC      $04DC00 ;  311      U_E8    DC      $00F400 ;   61
U_CB    DC      $049C00 ;  295      U_E9    DC      $00E400 ;   57
U_CC    DC      $045C00 ;  279      U_EA    DC      $00D400 ;   53
U_CD    DC      $041C00 ;  263      U_EB    DC      $00C400 ;   49
U_CE    DC      $03DC00 ;  247      U_EC    DC      $00B400 ;   45
U_CF    DC      $039C00 ;  231      U_ED    DC      $00A400 ;   41
U_D0    DC      $036C00 ;  219      U_EE    DC      $009400 ;   37
U_D1    DC      $034C00 ;  211      U_EF    DC      $008400 ;   33
U_D2    DC      $032C00 ;  203      U_F0    DC      $007800 ;   30
U_D3    DC      $030C00 ;  195      U_F1    DC      $007000 ;   28
U_D4    DC      $02EC00 ;  187      U_F2    DC      $006800 ;   26
U_D5    DC      $02CC00 ;  179      U_F3    DC      $006000 ;   24
U_D6    DC      $02AC00 ;  171      U_F4    DC      $005800 ;   22
U_D7    DC      $028C00 ;  163      U_F5    DC      $005000 ;   20
U_D8    DC      $026C00 ;  155      U_F6    DC      $004800 ;   18
U_D9    DC      $024C00 ;  147      U_F7    DC      $004000 ;   16
U_DA    DC      $022C00 ;  139      U_F8    DC      $003800 ;   14
U_DB    DC      $020C00 ;  131      U_F9    DC      $003000 ;   12
U_DC    DC      $01EC00 ;  123      U_FA    DC      $002800 ;   10
U_DD    DC      $01CC00 ;  115      U_FB    DC      $002000 ;    8
U_DE    DC      $01AC00 ;  107      U_FC    DC      $001800 ;    6
U_DF    DC      $018C00 ;   99      U_FD    DC      $001000 ;    4
U_E0    DC      $017400 ;   93      U_FE    DC      $000800 ;    2
U_E1    DC      $016400 ;   89      U_FF    DC      $000000 ;    0
U_E2    DC      $015400 ;   85
U_E3    DC      $014400 ;   81      ;
U_E4    DC      $013400 ;   77              ENDM
U_E5    DC      $012400 ;   73      ;
```

APPENDIX D

ALGLNXP.FOR

```
C     Generates a routine which
C     put in X:$180 the Alaw
C     coding table in the even
C     it inverted order
C
      DIMENSION
      IBUF(256),JBUF(256)
      OPEN(8,FILE='ALGLNXP.ASM')
C
      J=2
      IBUF(1)=-1
      IBUF(129)=1
      DO 10 I=2,62,2
      IBUF(J)=-(I+1)
      IBUF(128+J)=I+1
10    J=J+1
      DO 11 I=64,124,4
      IBUF(J)=-(I+2)
      IBUF(128+J)=I+2
11    J=J+1
      DO 12 I=128,248,8
      IBUF(J)=-(I+4)
      IBUF(128+J)=I+4
12    J=J+1
      DO 13 I=256,496,16
      IBUF(J)=-(I+8)
      IBUF(128+J)=I+8
13    J=J+1
      DO 14 I=512,992,32
      IBUF(J)=-(I+16)
      IBUF(128+J)=I+16
14    J=J+1
      DO 15 I=1024,1984,64
      IBUF(J)=-(I+32)
      IBUF(128+J)=I+32
15    J=J+1
      DO 16 I=2048,3968,128
      IBUF(J)=-(I+64)
      IBUF(128+J)=I+64
16    J=J+1
C
      CALL SCRAMBLE(IBUF,JBUF)
C
      WRITE(8,999)
999   FORMAT(
     1 '   opt    rc ,/,
     1 ';alglnxp.asm' ,/,
     1 ';this routine contents
       the half A-law decoding
       table ',/,';the table is
       in the even bit order and
       decodes ',/, ';directly
       the positive segments',/,
     1 ';it is located in the
       internal Y data memory
       ',/,';with the starting
       address X:180 hex.',//,
     1 '   org   X:$180',//';')
C
      DO 20 I=129,256
      IF(I.LT.17)WRITE(8,1000)
      I-1,ICONV(JBUF(I)),JBUF(I)
20    IF(I.GE.17)WRITE(8,1001)
      I-1,ICONV(JBUF(I)),JBUF(I)
1000  FORMAT('A_',Z1,'     DC
      $',Z6,' ;',I5)
1001  FORMAT('A_',Z2,'    DC
      $',Z6,' ;',I5)
      WRITE(8,1004)
1004  FORMAT(/'END',/,';')
      STOP 'end'
      END
      SUBROUTINE SCRAMBLE(IB,JB)
      DIMENSION IB(1),JB(1)
      DO 10 I=1,256
      J=IEOR(I-1,Z'55')
10    JB(J+1)=IB(I)
      END
      FUNCTION ICONV(I)
      ICONV=I*2048
      RETURN
      END
```

MLGLIN.FOR

```
C     MLGLIN.FOR
C
      DIMENSION IBUF(256)
C
      OPEN(8,FILE='MLOGLIN.ASM')
C
      J=1
      DO 10 I=1,29,2
      IBUF(128-J)=-(I+1)
      IBUF(256-J)=I+1
10    J=J+1
C
      DO 11 I=31,91,4
      IBUF(128-J)=-(I+2)
      IBUF(256-J)=I+2
11    J=J+1
C
      DO 12 I=95,215,8
      IBUF(128-J)=-(I+4)
      IBUF(256-J)=I+4
12    J=J+1
C
      DO 13 I=223,463,16
      IBUF(128-J)=-(I+8)
      IBUF(256-J)=I+8
13    J=J+1
C
      DO 14 I=479,959,32
      IBUF(128-J)=-(I+16)
      IBUF(256-J)=I+16
14    J=J+1
C
      DO 15 I=991,1951,64
      IBUF(128-J)=-(I+32)
      IBUF(256-J)=I+32
15    J=J+1
C
      DO 16 I=2015,3935,128
      IBUF(128-J)=-(I+64)
      IBUF(256-J)=I+64
16    J=J+1
      DO 17 I=4063,7903,256
      IBUF(128-J)=-(I+128)
      IBUF(256-J)=I+128
17    J=J+1
C
      WRITE(8,999)
999   FORMAT(';this macro
      contents the mu-law
      decoding table',/,
     1 ';it should be called
        with the starting
       address tab',/ /,
     1  'mloglin macro  tab',//
     1  '    org x:tab',//';')
C
      DO 20 I=1,256
      IF(I.LT.17)WRITE(8,1000)-
      I-1,ICONV(IBUF(I)),IBUF-
      (I)
20    IF(I.GE.17)WRITE(8,1001)-
      I-1,ICONV(IBUF(I)),IBUF-
      (I)
C
1000  FORMAT('U_',Z1,'     DC
       $',Z6,' ;',I5)
1001  FORMAT('U_',Z2,'    DC
       $',Z6,' ;',I5)
C
      WRITE(8,1004)
1004  FORMAT(/';',/,'
      ENDM',/,';')
      STOP 'end'
      END
C
      FUNCTION ICONV(I)
C
      ICONV=i*1024
      RETURN
      END
```

Literature Distribution Centres:
USA: Motorola Literature Distribution; P.O. Box 20912; Pheonix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands Milton Keynes, MK14 5BP, England.
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; P.O. Box 80300; Cheung Sha Wan Post Office; Kowloon Hong Kong.

MOTOROLA

Printed in Great Britain by Tavistock Press (Bedford) Ltd. 1000 2/90