1. Problem

This lab explores the details of interfacing the 68000 with analog-to-digital (A/D) and digital-to-analog (D/A) converters. Successive approximation, which makes use of a D/A converter to achieve A/D is thoroughly examined. The lab also explores companding.

While designing systems that interface with the real world, it is often necessary to get analog data from the real world *into* the computer for processing and similarly to bring digital data *out* of the computer back into analog form. These processes describe A/D and D/A conversion, respectively. This lab explores the hardware and software components and the processes used to accomplish such conversions.

Conversions are usually linear but there are instances in which there is significant benefit to non-linear conversions. Human hearing, for instance, has a logarithmic sensitivity scale and it makes sense to emphasize weak audio and suppress loud audio for listener comfort. In a compressor/expander (compander) arrangement, voice on a telephone line becomes more intelligible against noise because quiet signals become quantized in finer detail while louder signals have a coarse conversion with larger quantization steps. This lab examines the μ-law scheme for compressing audio input to provide relatively uniform “signal to quantizing noise ratio” over a range of input levels.

2. Design Specifications

A/D Converter:

An analog-to-digital converter must be able to sample an analog electrical signal (here, a voltage) and produce a corresponding digital code of n bits. This involves a three step process: (1) discretization of a continuous analog signal into exact samples at discrete times, (2) quantization of the analog signal into a finite number of states, and (3) encoding of the states into an n-bit digital code.

A linear analog-to-digital converter must provide a 1-to-1 mapping between any analog input level and post-quantization finite level. Additionally, a linear converter must have equal analog quantization steps Q such that:

$$Q = \frac{FS}{2^n}$$

Where FS= full scale range and n= number of bits used to represent binary output. The number of bits used should be selected such that the converter will register the smallest analog signal of interest and will have enough resolution to record the signal at the desired level of detail. As a general rule of thumb, n should be selected such that Q = level of noise in the analog signal.

In order to provide meaningful data conversion, the duration of sampling (aperture time) must be less than the amount of time it takes the analog signal to change by one Q. Also,

according to the Nyquist requirement, the A/D converter must sample the analog input at a rate of at least twice the highest frequency component. Mathematically:
$f_{sampling} > 2\ f_{cutoff}$

D/A Converter:

A digital-to-analog converter must be able to transform an n-bit binary code into an analog signal (voltage or current) in such a way that each discrete digital state results in one analog output. In other words, each digital state can map to a unique analog output or many digital states can map to the same analog output, but one digital state can not map to multiple analog outputs.

A D/A system uses the same quantization step identified earlier:

$$Q = \frac{FS}{2^n}$$

The specific situation may further require guaranteed monotonicity in the D/A's output. This means that for increasing digital inputs, the analog output must never decrease in value from the previous output. Guaranteed monotonicity is a characteristic of the D/A's hardware design.

Additional requirements specific to the application will determine: the analog output range of the D/A; whether it will include 0 as an output; and the maximum allowable settling time.

Companding System:

A compressing/expanding system (companding system) must be able to transmit a linear signal by compressing it at the sender, and expanding it at the receiver according to some agreed upon scheme. This scheme allows for non-linear A/D/A conversion so that small signals have a small Q (much detail) and large signals have a large Q (not as much detail necessary).

The goal of the compander is to provide a relatively uniform signal to quantizing noise ratio, S/Q. This results in consistent quality and intelligibility of a signal such as audio by allocating more bit space to low amplitude signals that would be easily drowned out by noise if converted using the same Q as a large amplitude signal.

The scheme used in the lab is the CCITT μ-Law. This is a logarithmic companding scheme (appropriate for audio) used in North America and Japan. It can be expressed by:

$$F(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)}$$

Where x is the normalized input signal (between -1 and +1), sgn(x) is the sign of x (1 for positive and 0 for negative), and μ is the compression parameter. Typically, μ=255. European networks use a similar scheme, called the A-law. Although implementation details differ, the basic idea of preserving the original signal to noise ratio is the same.

3. System Design

3.1 Hardware

Figure 1 shows the hardware as a block diagram. D/A conversion begins with the 555 timer, which strobes the PIA when it is time to perform a digital-to-analog conversion. This signal at the PIA results in an interrupt at the 68000 μP and the interrupt service routine begins to execute. The μP writes a byte to the PIA and this 8-bit value is output to the MC1408 D/A converter.

This particular D/A converter draws varying amounts of current through its I_O pin depending on the binary input, and so the MC741 op amp is used as an I/V converter to convert the current to an output voltage and also to buffer the output. The output analog waveform is now available and is fed back to the PIA in case successive approximation is to be used for A/D conversion.

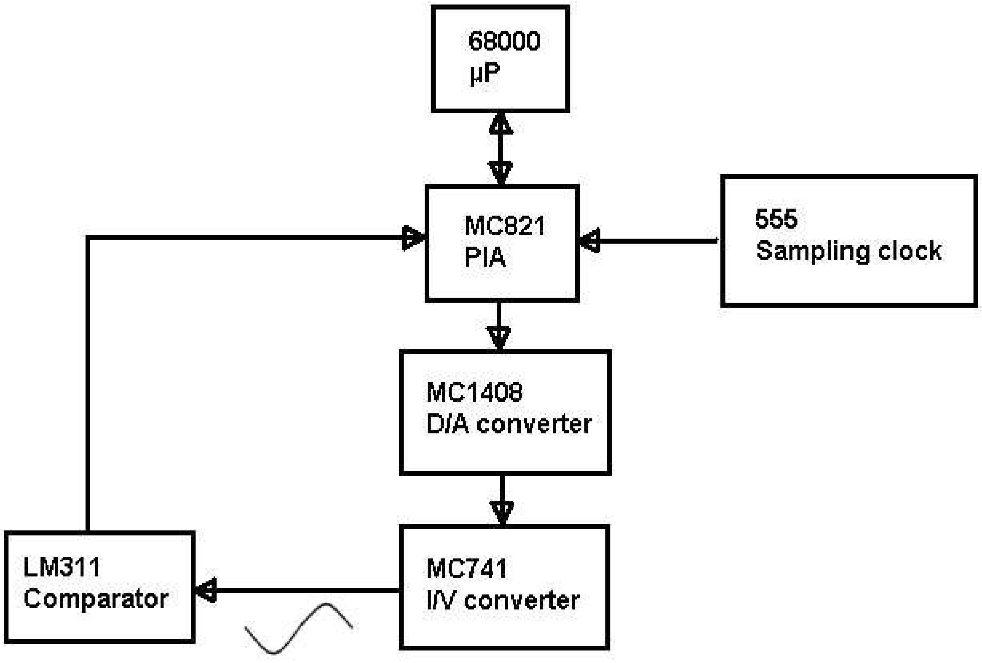

Figure 1: Hardware block diagram

3.2 Software

* Fully commented source code is included in the Appendix, at the end of the report.

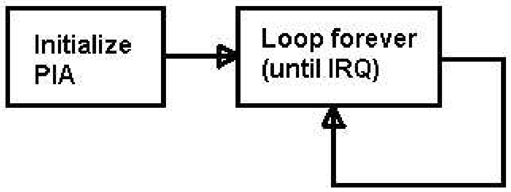

Figure 2: Program startup

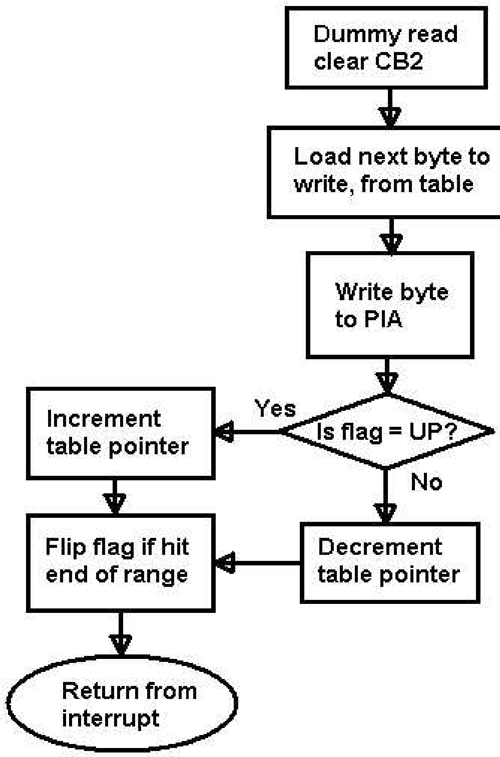

Figure 3: D/A ramp generator

a) Linear D/A Conversion

As shown in Figure 2, the program begins execution, and after initialization of the PIA it starts looping. The real work of the program is done by the interrupt service routine which is called when the Sampling clock (see Figure 1) strobes the PIA.

Figure 3 shows how the ramp generator simply reads values from a table and outputs them to the PIA. In the case of the linear D/A conversion, the table has been filled earlier with values ranging from 0x00 to 0xFF and so this program is writing increasing values from 0x00 up to 0xFF, and then back down. On the oscilloscope, the output of the D/A appears as a linear ramp because the voltages are increasing and decreasing by uniform steps.

b) Non-linear D/A Conversion

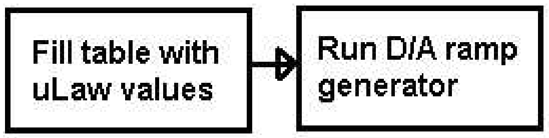

Figure 4: Non-linear D/A

This program has the exact same IRQ routine as the linear D/A converter. The only difference is that the table is filled with values from a μ-Law lookup table. This lets us achieve the desired non-linear effect or companding with no extra code.

c) Linear A/D Conversion

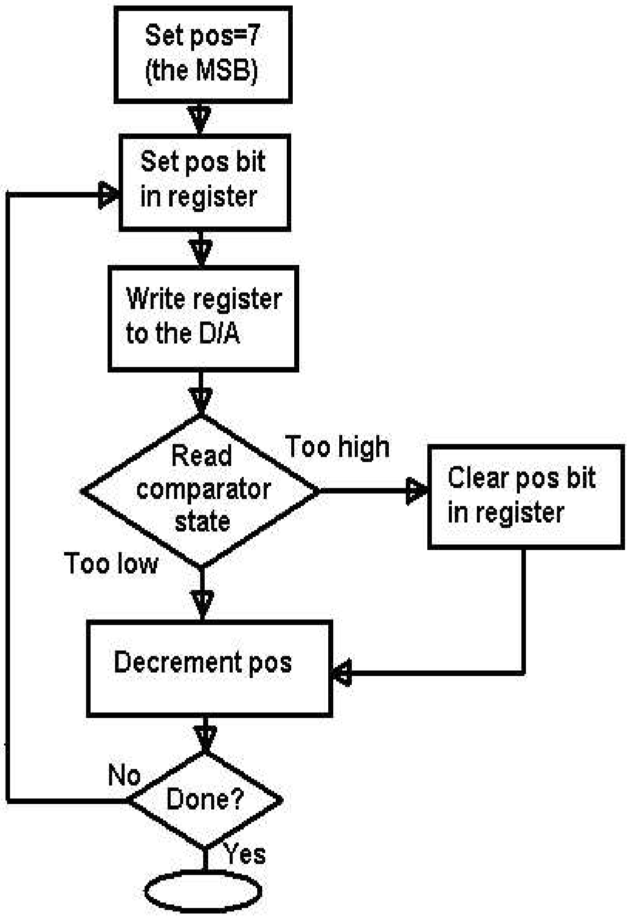

Figure 5: Successive approximation

We can build an A/D converter from a D/A converter and a comparator (see comparator feedback in Figure 1). The feedback makes successive approximation possible.

Successive approximation is a very efficient way to obtain a digital reading for an analog signal in few iterations. Figure 5 demonstrates how the process works. The software starts by guessing half of full range and writes the value to the D/A. The comparator provides feedback about whether the D/A produced a voltage that was too high or too low. If the guess was too high, then the last bit is cleared (otherwise it is kept as is). The process repeats until all bits are covered. On each iteration, half of the remaining range is covered and each iteration adds one extra bit of resolution.

d) *Non-Linear A/D Conversion*

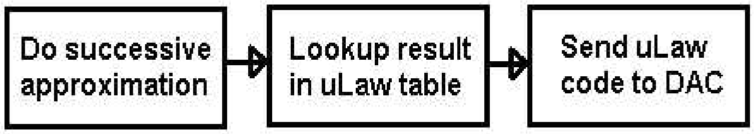

Figure 6: Non-linear A/D (with extra step for visualization)

This process is the same successive approximation as linear A/D. The only difference is that there is an extra step at the end to do a μ-Law lookup as shown in Figure 6. (In our lab, we also output the result to a 12-bit DAC for viewing). Please see the source for NAD.ASM in Section 8, Appendix.

4. System Implementation

The functionality described in Section 3 was implemented according to Table 1.

Component	**Functionality**
Rb	The sampling clock's frequency is determined by an RC circuit. Rb defines the frequency of the timer and is a variable resistor.
LM555	The 555 is a popular timer. Here it is used to generate a clock signal to interrupt the microprocessor via the PIA. It is our sampling clock.
R1	The reference amplifier current flows into pin 14 of the D/A. R1 determines how much current flows into the D/A and hence determines the maximum output current. D/A's Io = (Vref/R1) x (input / 255)
MC1408	The D/A converter itself. This is a current reference amplifier that uses an R-2R ladder and 8 high speed current switches. It reads an 8-bit digital value and draws an appropriate amount of current from Io.
R3	This is the feedback resistor in the op-amp circuit. It is used to convert a current to a voltage.
MC741	The operational amplifier. It acts as a current to voltage converter.
LM311	A common comparator. It compares two analog voltages and produces HIGH output if the trial voltage is higher than the reference. Vital to successive approximation in the feedback circuit to the PIA.

Table 1: Implementation

5. Results and Discussion

Sketches of Waveforms

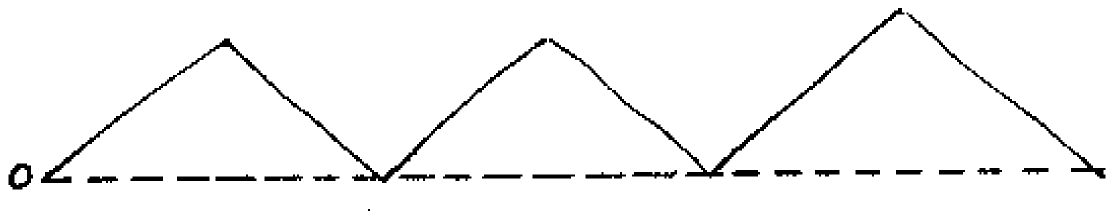

Figure 7: Linear D/A, output ramp

Figure 7 shows the output of the simple linear digital-to-analog converter. Increasing values were read from the table until the end of the table, then decreasing values were read. By sending each of the values out to the DAC we get this nice, linear output.

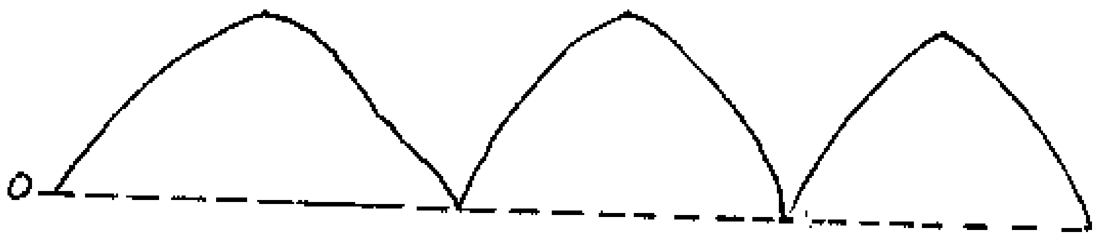

Figure 8: Non-linear D/A, output "ramp"

For non-linear conversion, we used essentially the same program as the linear conversion but did a µ-Law lookup before outputting the value to the DAC. In Figure 8 we can see the effect of the companding. For low input values (near 0), the output changes rapidly because µ-Law is giving an emphasis to the otherwise easy-to-lose-in-noise small signal.

For larger input values, the output changes very slowly. Large signals don't need as much "detail" since they are already much greater than noise.

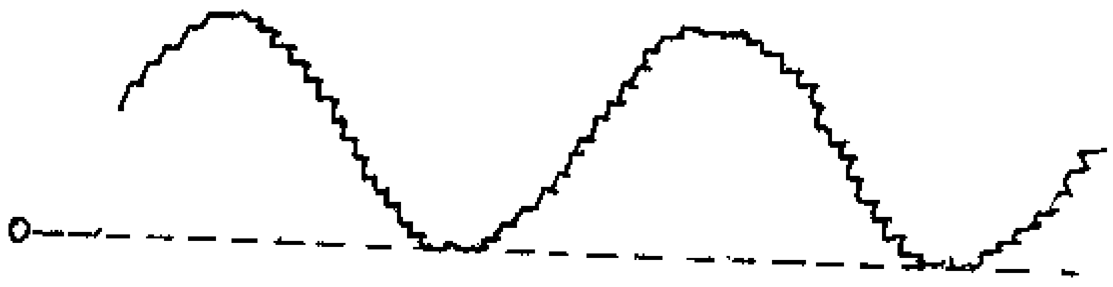

Figure 9: Linear A/D, successive approximation

Figure 9 shows successive approximation in action. The binary decision circuit (implemented in software) attempts to locate the signal by making a binary decision (up or down) on each iteration. It is able to rapidly determine the signal to a high resolution.

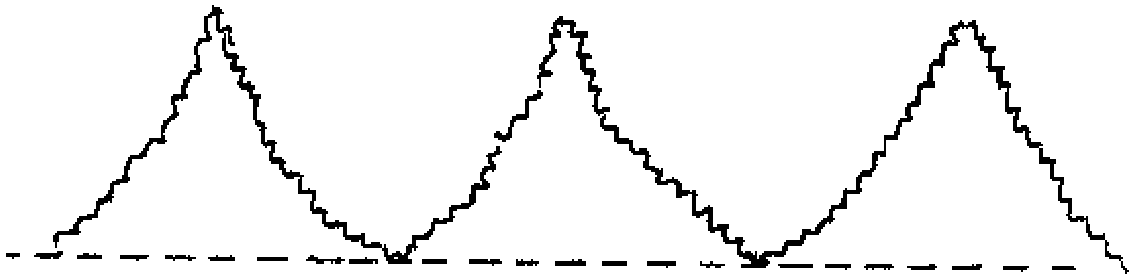

Figure 10: Non-linear A/D, the DAC output

Figure 10 shows the effects of successive approximation combined with a µ-Law lookup. The jagged nature of the signal is a result of successive approximation and the compression of the signal is due to the µ-Law conversion.

Analysis of Interrupt Service Routine time

The analog-to-digital ISR took approximately 1000 µs to execute. While the ¬IRQ is low our ISR is being executed, and when ¬IRQ returns high the service routine is completed. Because we observed that ¬IRQ spent very little time in the high state we can surmise that the processor was very close to interrupt saturation. If we observed just brief +ve spikes, we would know that the processor was experiencing interrupt saturation.

The sampling rate was variable. The variable resistor by the 555 timer let us adjust the rate at which the microprocessor was interrupted. The maximum sampling rate supported would be the point at which interrupt saturation just occurs. Any further than that, and we would be missing samples.

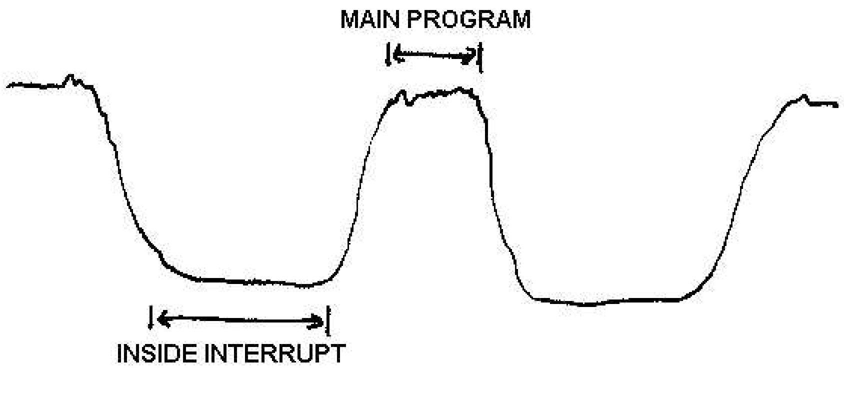

Figure 11: ¬IRQ pin

6. Points to Ponder

1.

The Nyquist criterion establishes a general rule for analog signal digitization. It states that for proper signal conversion, the sampling frequency (f_s) must be at least double the cut-off frequency (f_c) of the signal. Another way of thinking of this is that the sampling frequency must exceed twice the highest frequency component in the signal:
$f_{sampling} > 2\ f_{cutoff}$

For instance, humans can perceive audio from about 20 Hz to 22 KHz (if they're lucky). Therefore it makes sense for modern PC sound cards to sample audio at a rate of 44 KHz, or twice the highest frequency component that's worthwhile.

2.

The aperture time is the amount of time the conversion takes to happen. It's related to conversion speed. For the conversion to be valid, the signal must not change more than one Q during the aperture time. For this reason a smaller aperture time is often desired.

3.

The purpose of companding is to increase intelligibility (or integrity) of a signal that's vulnerable to noise while using the same allotted bandwidth: it is a compressing/expanding system. This is achieved by boosting the low amplitude signals (via a smaller Q) and reducing the resolution of high amplitude signals (larger Q). This works because a high amplitude signal is already immune to noise whereas an unaided low amplitude signal is in danger of being lost in noise.

By using a non-linear companding system, we can achieve a relatively uniform signal to quantizing noise ratio, S/Q across the range of signals. In a linear system, on the other hand, the S/Q ratio is proportional to the signal level.

4.

Dynamic range is defined as: $10 \cdot \log_{10} \frac{P_{FS}}{P_Q}$ where P_{FS} is the power of the full scale analog signal, and P_Q is the power of the quantization step Q. The more bits you have to represent the signal, the larger the dynamic range. Dynamic range is related to the range between the quietest and loudest signals. Because companding boosts the quietest signals, it increases the dynamic range and a better signal to noise ratio is achieved.

5.

The MC1408 is designed to give output as current, which is a linear product of an 8-bit value and a reference voltage. It uses an R-2R ladder, current amplifier and current switches, and the nature of the device is that it deals with currents and not voltages. The current steering is also inherently fast.

6.

Two MC1408 can be combined to produce “16 bits of output”, however they can not be used to construct a converter that’s accurate to 16 bits. The product data sheet also suggests that two 8-bit converts SHOULD NOT be used to construct a 16-bit unit.

A single MC1408 guarantees accuracy to ± ½ LSB. That’s 0.5 in 256 or ± 0.19%. So we’ll never do better than that using the MC1408.

However, a 16-bit converter with ± ½ LSB accuracy means that we need a maximum error of 0.5 in 65536, or ± 0.00076%. Clearly, there is no way to achieve this using the MC1408.

7.

The main advantage of a successive approximation converter is the high speed of conversion (deterministic time, number of steps determined by bit size). The other main advantage is that the converter can easily be extended to more bits for higher resolution. It’s also very binary friendly and thus easy to implement in hardware or software.

8.

The bottleneck depends on the feedback resistor used with op-amp. If the slew rates were equal, 8mA * R = 0.5 and so R = 62.5 Ω. This is clearly much lower a value then we would use in practice for such a circuit. Therefore the MC1408 is likely to be the bottleneck.

9.

In the lab, the 555 timer controlled the sampling rate. We were able to adjust it via a variable resistor (potentiometer) which is labeled Rb in the circuit diagram. This variable resistor could be replaced with another unit with a larger range if we wanted a wider range of clock frequencies.

For an input sine wave of 2V peak to peak with 1 KHz frequency we have to consider the aperture time. We must sample the signal fast enough so that the signal does not change by one Q during the sampling.

What’s one quantization step? Here it’s $2 / 2^8 = 1 / 128$ of a volt. And the maximum slope of the signal is $2\pi f$. If we make a triangle on the signal we can see that slope = rise / run and therefore $2\pi f = Q / t$.

The solution to this is that $t = Q / 2\pi f = (1/128) / (6283) =$ **1.24 µs**. In the worst case, we have to convert it faster than this to ensure that the signal doesn’t change by one Q.

10.

Successive approximation is not the optimal technique for a signal that changes very slowly, like a temperature. A more adequate conversion technique for such a signal would be a tracking type A/D. This is because once it is tracked on the signal, it will convert very rapidly: conversion time equals clock rate. It’s unlikely that there will be discontinuities (rapid jumps) in the temperature!

For sampling voice or other audio (many different frequency components) successive approximation is superior. We can not track a signal like that because we would immediately lose tracking. Successive approximation will locate the signal quickly each time.

7. References

Audio-Technica U.S., Inc., *Companding*. Public World Wide Web Document, July 2001 http://www.audio-technica.com/using/wireless/advanced/compand.html

W. Kinsner, *Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems.* Lecture Notes, September 2002 (v7.33)

8. Appendix: Assembly Source

LDA.ASM	Linear digital-to-analog conversion
NDA.ASM	Non-linear digital-to-analog conversion
LAD.ASM	Linear analog-to-digital conversion
NAD.ASM	Non-linear analog-to-digital conversion

```
** 24.424 Microprocessor Interfacing
** Lab 3 (LDA.ASM)
**
** Name: Jem Berkes
** Date: 2002-10-11
**
*************************************************************************
**                          Memory Defs and Constants
*************************************************************************
PORTA   EQU     $30039          * LOCATION OF PIA PORT A
DRA     EQU     $30039          * LOCATION OF PIA DIRECTION REGISTER A
CRA     EQU     $3003B          * LOCATION OF PIA CONTROL REGISTER A
PORTB   EQU     $3003D          * LOCATION OF PIA PORT B
DRB     EQU     $3003D          * LOCATION OF PIA DIRECTION REGISTER B
CRB     EQU     $3003F          * LOCATION OF PIA CONTROL REGISTER B
TABLE   EQU     $3000           * LOCATION OF D/A LOOK UP TABLE


*************************************************************************
**                          Stack Area
*************************************************************************
        ORG     $7000
S_STACK DC.B    1


*************************************************************************
**                              MAIN PROGRAM
*************************************************************************
        ORG     $1000                 * Start Program at 0x1000
        LEA     S_STACK,SP            * Load Stack Pointer with 0x7000
        ORI.W   #$0700,SR             * Disable Interupts
        MOVE.L  #$4000,$00000070     * Load Interupt Routine Pointer Into
*                                       IRQ Vector
        JSR     PIAINIT               * Init PIA
        JSR     TBLINIT
        ANDI.W  #$FBFF,SR             * Enable Interupts
waitIRQ BRA     waitIRQ               * Loop And Wait Until An Interrupt
*                                       Request


*************************************************************************
**                              PIAINIT ROUTINE
**   Initalize the PIA.
*************************************************************************
PIAINIT MOVE.B  #$00,CRA              * Enable Direction Register A
        MOVE.B  #$00,PORTA            * Set Port A to Input
        MOVE.B  #$04,CRA              * Enable Data Register A

        MOVE.B  #$00,CRB              * Enable Direction Register B
        MOVE.B  #$FF,PORTB            * Set Port B to Output
        MOVE.B  #$0C,CRB              * Enable Data Register B,
*                                      Enable Interrupt on CB2 Transition
        RTS


*************************************************************************
**                              TBLINIT ROUTINE
**
```

```
**   This routine initalizes the table located at location TABLE with
**
**   the appropriate values for the LDA look up table. The table should be
**
**   a ramp going from $00 to $FF.
**
***************************************************************************
****
TBLINIT LEA.L   TABLE,A0
        CLR.L   D0
TBLLOOP MOVE.B  D0,(A0)+
        ADDI.B  #$01,D0
        BNE     TBLLOOP
        CLR.L   D4     * UP/DOWN IDENTIFIER (zero = up)
        CLR.L   D5     * position counter
        RTS

***************************************************************************
**                              INTERRUPT ROUTINE
**
**   This routine must output the appropriate TABLE value to port B. The
**
**   routine must keep track of whether or not it is going up, or down,
**
**   and which is the current position in the table. The interrupt routine
**
**   MUST also clear the CB2 flag as to not automatically return to the
**
**   interrupt routine.
**
***************************************************************************
****

IRQROUT ORG     $4000
            MOVE.B    PORTB,D2       * Dummy read of PORTB to clear CB2
            LEA.L     TABLE,A1       * Point to start of table
            ADDA.L    D5,A1          * Point (count) positions into table
            MOVE.B    (A1),PORTB     * Write byte at (table pointer) to PORTB
            TST       D4             * Test up/down identifier
            BEQ       UPWARD         * incrementing
            SUBQ.B    #1,D5          * decrement position
            BEQ       FLIPIT         * hit zero, so flip direction
            RTE
UPWARD      CMPI.B    #$FF,D5        * if offset is not FF
            BNE       GO_UP          * go increment
            SUBQ.B    #1,D5          * else decrement
            BRA       FLIPIT         * and flip direction
GO_UP       ADDQ.B    #1,D5
            RTE
FLIPIT      NOT       D4             * flip the up/down counter
            RTE

            END
```

```
** 24.424 Microprocessor Interfacing
** Lab 3 (NDA.ASM)
**
** Name: Jem Berkes
** Date: 2002-10-11
**
*************************************************************************
**                          Memory Defs and Constants
**
*************************************************************************
****
PORTA   EQU     $30039          * LOCATION OF PIA PORT A
DRA     EQU     $30039          * LOCATION OF PIA DIRECTION REGISTER A
CRA     EQU     $3003B          * LOCATION OF PIA CONTROL REGISTER A
PORTB   EQU     $3003D          * LOCATION OF PIA PORT B
DRB     EQU     $3003D          * LOCATION OF PIA DIRECTION REGISTER B
CRB     EQU     $3003F          * LOCATION OF PIA CONTROL REGISTER B
TABLE   EQU     $3000           * LOCATION OF D/A LOOK UP TABLE


*************************************************************************
**                          Stack Area
**
*************************************************************************
****
        ORG     $7000
S_STACK DC.B    1


*************************************************************************
**                              MAIN PROGRAM
**
*************************************************************************
****
        ORG     $1000                 * Start Program at 0x1000
        LEA     S_STACK,SP            * Load Stack Pointer with 0x7000
        ORI.W   #$0700,SR             * Disable Interupts
       MOVE.L  #$4000,$00000070      * Load Interupt Routine Pointer Into
*                                       IRQ Vector
        JSR     PIAINIT               * Init PIA
        CLR.L   D4    * UP/DOWN IDENTIFIER (zero = up)
        CLR.L   D5    * position counter
        ANDI.W  #$FBFF,SR             * Enable Interrupts
waitIRQ BRA     waitIRQ               * Loop And Wait Until An Interrupt
*                                       Request


*************************************************************************
**                              PIAINIT ROUTINE
**
**   Initalize the PIA.
**
*************************************************************************
****
PIAINIT MOVE.B  #$00,CRA              * Enable Direction Register A
        MOVE.B  #$00,PORTA            * Set Port A to Input
        MOVE.B  #$04,CRA              * Enable Data Register A
        MOVE.B  #$00,CRB            * Enable Direction Register B
        MOVE.B  #$FF,PORTB          * Set Port B to Output
        MOVE.B  #$0C,CRB            * Enable Data Register B,
*                                    Enable Interrupt on CB2 Transition
        RTS


*************************************************************************
**                              INTERRUPT ROUTINE
**
**   This routine must output the appropriate TABLE value to port B. The
**
**   routine must keep track of whether or not it is going up, or down,
**
**   and which is the current position in the table. The interrupt routine
**
**   MUST also clear the CB2 flag as to not automatically return to the
**
**   interrupt routine.
*************************************************************************
****

IRQROUT ORG     $4000
            MOVE.B    PORTB,D2       * Dummy read of PORTB to clear CB2
            LEA.L     TABLE,A1       * Point to start of table
            ADDA.L    D5,A1          * Point (count) positions into table
            MOVE.B    (A1),PORTB     * Write byte at (table pointer) to PORTB
            TST       D4             * Test up/down identifier
            BEQ       UPWARD         * incrementing
            SUBQ.B    #1,D5          * decrement position
            BEQ       FLIPIT         * hit zero, so flip direction
            RTE
UPWARD      CMPI.B    #$FF,D5        * if offset is not FF
            BNE       GO_UP          * go increment
            SUBQ.B    #1,D5          * else decrement
            BRA       FLIPIT         * and flip direction
GO_UP       ADDQ.B    #1,D5
            RTE
FLIPIT      NOT       D4             * flip the up/down counter
            RTE

          ORG TABLE
            µ-Law table follows

            END
```

```
** 24.424 Microprocessor Interfacing
** Lab 3 (LAD.ASM)
**
** Name: Jem Berkes
** Date: 2002-10-11
**
*************************************************************************
**                           Memory Defs and Constants
**
*************************************************************************
****
PORTA   EQU     $30039          * LOCATION OF PIA PORT A
DRA     EQU     $30039          * LOCATION OF PIA DIRECTION REGISTER A
CRA     EQU     $3003B          * LOCATION OF PIA CONTROL REGISTER A
PORTB   EQU     $3003D          * LOCATION OF PIA PORT B
DRB     EQU     $3003D          * LOCATION OF PIA DIRECTION REGISTER B
CRB     EQU     $3003F          * LOCATION OF PIA CONTROL REGISTER B
TABLE   EQU     $3000           * LOCATION OF D/A LOOK UP TABLE


*************************************************************************
**                           Stack Area
**
*************************************************************************
****
        ORG     $7000
S_STACK DC.B    1


*************************************************************************
**                               MAIN PROGRAM
**
*************************************************************************
****
        ORG     $1000                 * Start Program at 0x1000
        LEA     S_STACK,SP            * Load Stack Pointer with 0x7000
        ORI.W   #$0700,SR             * Disable Interupts
        MOVE.L  #$4000,$00000070      * Load Interupt Routine Pointer Into
*                                       IRQ Vector
        JSR     PIAINIT               * Init PIA
        ANDI.W  #$FBFF,SR             * Enable Interrupts
waitIRQ BRA     waitIRQ               * Loop And Wait Until An Interrupt
*                                       Request


*************************************************************************
**                               PIAINIT ROUTINE
**
**   Initalize the PIA.
**
*************************************************************************
****
PIAINIT MOVE.B  #$00,CRA              * Enable Direction Register A
        MOVE.B  #$00,PORTA            * Set Port A to Input
        MOVE.B  #$04,CRA              * Enable Data Register A

        MOVE.B  #$00,CRB              * Enable Direction Register B
        MOVE.B  #$FF,PORTB            * Set Port B to Output
        MOVE.B  #$0C,CRB                 * Enable Data Register B,
*                                         Enable Interrupt on CB2 Transition
        RTS


************************************************************************
**                                INTERRUPT ROUTINE
**
**    This interrupt must perform a linear analog to digital conversion on
**
**    the signal appearing on J8 (PIN 3) of the LM311 comparater. The
**
**    routine should perform a successive approximation on the signal.
 **
************************************************************************
****
IRQROUT ORG     $4000
        CLR.B   D3
        MOVE.B  #7,D0          * Set D0=7 (highest bit first)
LOOP    BSET.B  D0,D3          * Send this bit through DAC
        MOVE.B  D3,PORTB
        BTST    #7,PORTA       * Test the result from comparator
        BEQ     PROCEED        * Too low; keep this bit
        BCLR.B  D0,D3          * Too high; clear this bit
PROCEED SUBQ.B  #1,D0
        BNE     LOOP           * Try next MSB
        MOVE.B  PORTB,D1       * Dummy read to clear CB2
        RTE

        END
```

```
** 24.424 Microprocessor Interfacing
** Lab 3 (NAD.ASM)
**
** Name: Jem Berkes
** Date: 2002-10-11
**
*************************************************************************
**                           Memory Defs and Constants
**
*************************************************************************
****
DAC     EQU     $20000          * LOCATION OF 12-BIT DAC
PORTA   EQU     $30039          * LOCATION OF PIA PORT A
DRA     EQU     $30039          * LOCATION OF PIA DIRECTION REGISTER A
CRA     EQU     $3003B          * LOCATION OF PIA CONTROL REGISTER A
PORTB   EQU     $3003D          * LOCATION OF PIA PORT B
DRB     EQU     $3003D          * LOCATION OF PIA DIRECTION REGISTER B
CRB     EQU     $3003F          * LOCATION OF PIA CONTROL REGISTER B
TABLE   EQU     $3000           * LOCATION OF D/A LOOK UP TABLE


*************************************************************************
**                           Stack Area
**
*************************************************************************
****
        ORG     $7000
S_STACK DC.B    1


*************************************************************************
**                               MAIN PROGRAM
**
*************************************************************************
****
        ORG     $1000                 * Start Program at 0x1000
        LEA     S_STACK,SP            * Load Stack Pointer with 0x7000
        ORI.W   #$0700,SR             * Disable Interupts
       MOVE.L  #$4000,$00000070      * Load Interupt Routine Pointer Into
*                                       IRQ Vector
        JSR     PIAINIT               * Init PIA

        ANDI.W  #$FBFF,SR             * Enable Interrupts
waitIRQ BRA     waitIRQ               * Loop And Wait Until An Interrupt
*                                       Request


*************************************************************************
**                               PIAINIT ROUTINE
**
**   Initalize the PIA.
**
*************************************************************************
****
PIAINIT MOVE.B  #$00,CRA              * Enable Direction Register A
        MOVE.B  #$00,PORTA            * Set Port A to Input
        MOVE.B  #$04,CRA              * Enable Data Register A
        MOVE.B  #$00,CRB              * Enable Direction Register B
        MOVE.B  #$FF,PORTB            * Set Port B to Output
        MOVE.B  #$0C,CRB              * Enable Data Register B,
*                                      Enable Interrupt on CB2 Transition
        RTS


*************************************************************************
**                               INTERRUPT ROUTINE
**
**    This interrupt must perform a linear analog to digital conversion on
**
**    the signal appearing on J8 (PIN 3) of the LM311 comparater. The
**
**    routine should perform a successive approximation on the signal.
*************************************************************************
****
IRQROUT ORG     $4000
        CLR.L   D3              * D3 will hold result
        MOVE.B  #7,D0           * Set D0=7 (highest bit first)
LOOP    BSET.B  D0,D3           * Set this bit in our result
        MOVE.B  D3,PORTB        * Send this result to DAC
        BTST    #7,PORTA        * Test the comparator's response
        BEQ     PROCEED         * if ZERO, keep the bit
        BCLR.B  D0,D3           * else clear bit
PROCEED SUBQ.B  #1,D0           * point to next MSB
        BNE     LOOP
        LEA     TABLE,A0        * Point A0 to table
        ADDA.L  D3,A0           * A0 points to table+(code)
        CLR.L   D5
        MOVE.B  (A0),D5         * D5 contains uLaw result
        LSL.L   #4,D5           * Shift result left by 4 bits
        MOVE.W  D5,DAC          * Write uLaw result to 12-bit DAC
        MOVE.B  PORTB,D1        * Dummy read to clear CB2
        RTE

          ORG TABLE
            µ-Law table follows

        END
```