# MICRO-CONTROLLER AND DIGITAL SIGNAL PROCESSING

An investigation into digital signal processing,

with emphasis on simplicity and practically.

by

Seyed Akhavi

Submitted in Partial Fulfillment of the Requirements

for the degree of Master of Science in Engineering

in the

Electrical Engineering

Program

Professor Samuel J. Skaroti 4/12/92

Advisor Date

Sally M. Hotchkiss June 15, 1992

Dean of the Graduate School Date

YOUNGSTOWN STATE UNIVERSITY

June, 1992

ABSTRACT

# MICRO-CONTROLLER AND DIGITAL SIGNAL PROCESSING

SEYED AKHAVI

MASTER OF SCIENCE IN ENGINEERING

YOUNGSTOWN STATE UNIVERSITY, 1992

New advances in digital electronics have made it possible to manufacture micro-controllers which are capable of analyzing and processing signals. Now applications that require bulky components and complex hardware can be implemented by using microcontrollers. The auto industry is using these single chips to monitor and control the speed of the motor, fuel injection rate and spark timing. Precision electronic instruments, peripheral devices, communication devices such as pagers, laser printers, color copiers, are all equipped with one or more microcontroller chips. A modern approach, which takes full advantage of the microcontroller's programmability, has made it feasible to simulate such tasks as filtering in the digital domain. This thesis investigates the design and implementation of a 5-band audio signal analyzer by using the latest 16-bit microcontroller manufactured by Motorola. Methods and techniques involved in the areas of analog-to-digital conversion, digital signal processing, serial interfacing, and related programming routines are discussed and developed. Hardware as well as software design goals, with the emphasis on programmability of microcontrollers, are presented.

# ACKNOWLEDGMENTS

I wish to acknowledge the assistance of my advisor Professor Skarote for teaching the courses which gave birth to the ideas of this project. His teachings have always been an inspiration for me to try and pursue new ideas.

I also wish to thank Microcontroller Unit of Motorola for their assistance in obtaining literature, samples, and information. And finally, I must acknowledge my deep debt to my wife, Mansureh, without whose support, patience, and tolerance I would not have brought this project to conclusion.

# TABLE OF CONTENTS

# LIST OF SYMBOLS

| SYMBOL | DEFINITION | UNITS OR REFERENCE |
|---|---|---|
| ADC | Analog-to-digital conversion | |
| ABIU | A bus interface unit | |
| ADCMCR | ADC module configure register | |
| ADCTL0 | ADC control register 0 | |
| ADCTL1 | ADC control register 1 | |
| ADSTAT | ADC status Register | |
| AFA | Audio frequency analyzer | |
| ALU | Arithmetic logic unit | |
| bit | Binary digit 0 or 1 | |
| byte | Group of bits, usually eight | |
| C | Capacitance | farad |
| CA:CD | Channel selection field | |
| CCF | Conversion complete field | |
| CCTR | Conversion counter field | |
| CPU | Central processing unit | |
| DAC | Digital to analog conversion | |
| DSP | Digital signal processing | |
| dB | Decibel, unit of logarithmic power ratio | none |
| Fc | Center frequency | cycles/second |
| $F_S$ | Sampled frequency | cycles/second |
| FRZ | Freeze | |
| GPT | General purpose timer | |

| | | |
|---|---|---|
| HCMOS | High-density complementary metal oxide semiconductor | |
| I/O | Input/output | |
| IIR | Infinite impulse response | |
| IMB | Intermodule bus | |
| kbyte | kilo- (one thousand byte) | |
| kHz | kilo- (one thousand hertz) | |
| L | Inductance | henry |
| LED | Light emitting diode | |
| Mag | Magnitude | dB |
| MCU | Microcontroller unit | |
| MULT | Multichannel conversion bit | |
| MSB | Most significant bit | |
| PIT | Programmable interrupt timer | |
| PRS | Prescalar rate selection | |
| Q | Quality factor | none |
| QSM | Queued serial module | |
| QSPI | Queued serial peripheral interface | |
| R | Resistance | ohm |
| RAM | Random access memory | |
| RC | Resistor-capacitor | |
| Ref. | Reference | |
| Res. | Resolution | |
| rms | Root mean square | |
| ROM | Read only memory | |

| | | |
|---|---|---|
| S8CM | Select eight-conversion sequence mode | |
| SAA | Small angle approximation | |
| SAR | Successive approximation register | |
| SCAN | Scan mode selection bit | |
| SCF | Sequence complete flag | |
| SCI | Serial communication interface | |
| SIM | System integration module | |
| SRAM | Standby RAM | |
| STS | Sample time select field | |
| SUPV | Supervisor/unrestricted | |
| Tc | Clock period | second |
| Ts | Sampled period | second |
| VLSI | Very large scale integration | |
| $Z^{-1}$ | Inverse z-transform operator | |
| $\phi$ | Phase angle | radian |
| $\theta$ | Sampled period | radian |
| $\alpha$ | Coefficient related to analog domain in z | |
| $\beta$ | Coefficient related to analog domain in z | |
| $\gamma$ | Coefficient related to analog domain in z | |
| $A(\omega)$ | Gain | |

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

## OVERVIEW

Semiconductor manufacturers have produced 16- and 32- bit microprocessors which can handle larger sizes of data at a faster rate. These microprocessors not only are used in fast computers and workstations but also they have been employed in real-time control applications. The microprocessor, and several other chips which handle timing, interrupts, serial communications, and parallel communications might be needed in design of a real-time control. In a real time-control the microprocessor interacts with resources which make up an instrument or device, at times dictated by the hardware external to the microprocessor. Many applications require fast response and service by a microprocessor. One approach that semiconductor manufacturers have chosen to increase the system speed is to build faster microprocessors. Another technique has combined the microprocessor with RAM, ROM, and variety of input/output capabilities on a single chip, which is commonly referred to as a microcontroller.

This technique has been made possible by recent advances in very large scale integration (VLSI) chip manufacturing. Now it is conceivable to build a single chip that contains all of the major elements of a computer system. This single chip, which is a microcontroller unit (MCU), includes the central processing unit (CPU), comprised of the arithmetic logic unit (ALU) and control unit, plus interfacing devices and memory units. The MCU is a single chip which is built up from standard modules that interface via a common internal bus[1]. The study of a 16-bit microcontroller and its application in digital signal processing (DSP) is the main scope of this thesis. The

advantages of the digital domain over the analog domain have initiated a new trend where traditional electronic circuits are now being emulated in software and hardware. One such area is in digital signal processing. This new method offers a primary advantage over analog and has much greater flexibility. The programmability of DSP is what makes it flexible. If a designer is using analog components such as resistors and capacitors in a low pass filter, and later on might want to make changes to the filter specifications, one must go back and replace the wiring as well as the components. Once DSP is applied to make any changes, the designer need only substitute some of the codes in the software and in this way alter the behavior of the filter. Fast microcontrollers have made it possible to generate and process frequencies up to 40 kHz.

The designed project uses a microcontroller to:

- Convert the audio signal into a digital signal.
- Sample and subject the data signal to five DSP band pass filter algorithms.
- Extract the peak of each filter.
- Encode the peaks of each band and send them through queued serial peripheral interface (QSPI) to the LED array in real time.

Figure 1 in the next page shows the block diagram of the project.

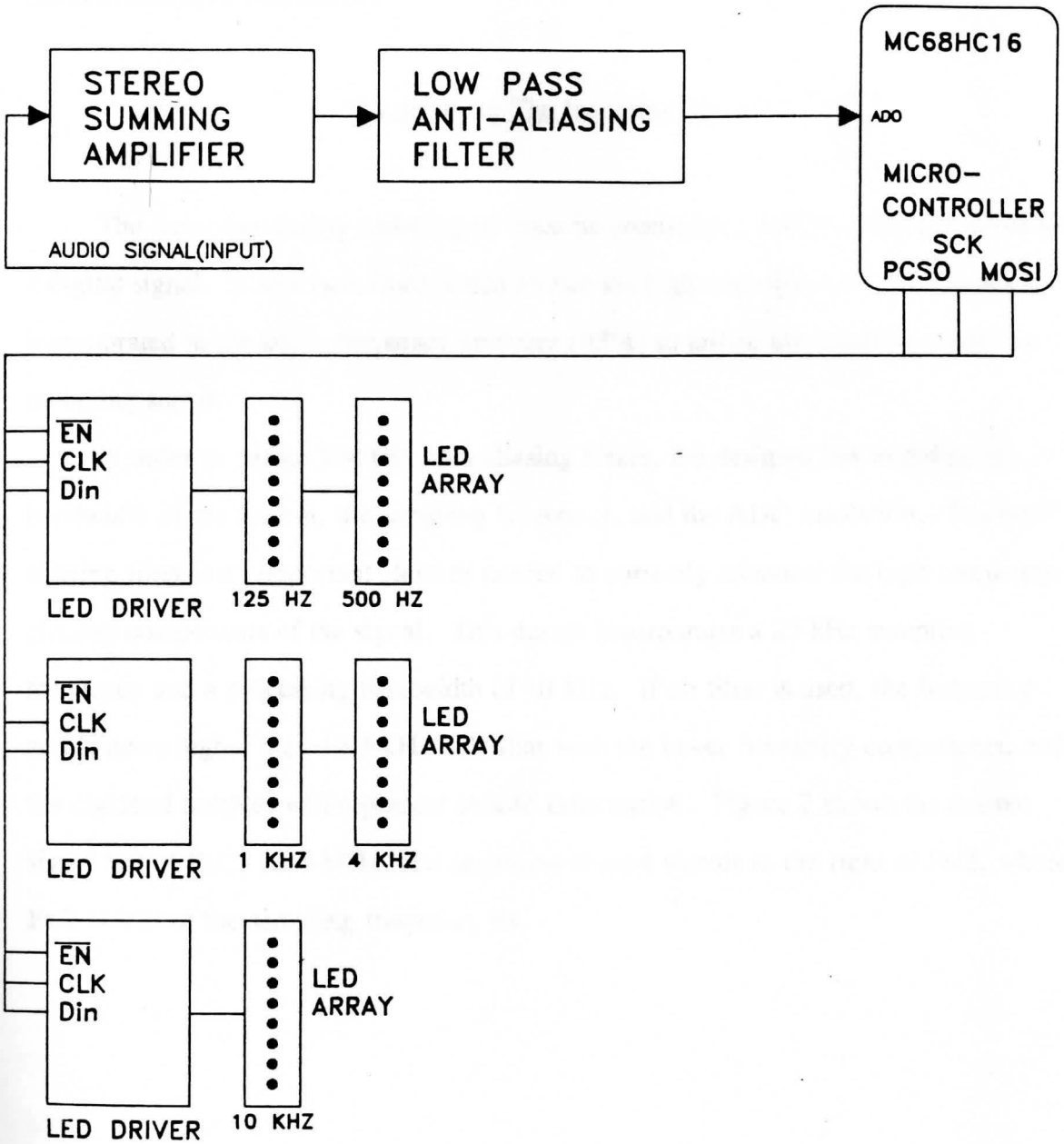



Figure 1. Audio Frequency Analyzer System Diagram

# SELECTION OF DESIGN CRITERIA

## DESIGN SPECIFICATIONS

### Conditioning The Incoming Signal

The incoming analog audio signal must be conditioned before being converted to a digital signal. A low pass filter which eliminates high and unwanted frequencies is incorporated in the audio frequency analyzer (AFA) to utilize the conditioning of the incoming signal.

In order to design low pass anti-aliasing filters, the designer has to define the bandwidth of the system, the sampling frequency, and the ADC resolution. The anti-aliasing filter is an important element needed to correctly attenuate the high frequency aliasing components of the signal. This design incorporates a 25 kHz sampling frequency and a processing bandwidth of 10 kHz. If no filter is used, the frequency components higher than 12.5 kHz will alias with the lower frequency components, and the digitized samples will represent invalid information. Figure 2 shows the desired signal left of Fs/2, 12.5 MHz, and undesired aliased signals to the right of Fs/2, where Fs/2 is half of the sampling frequency Fs.

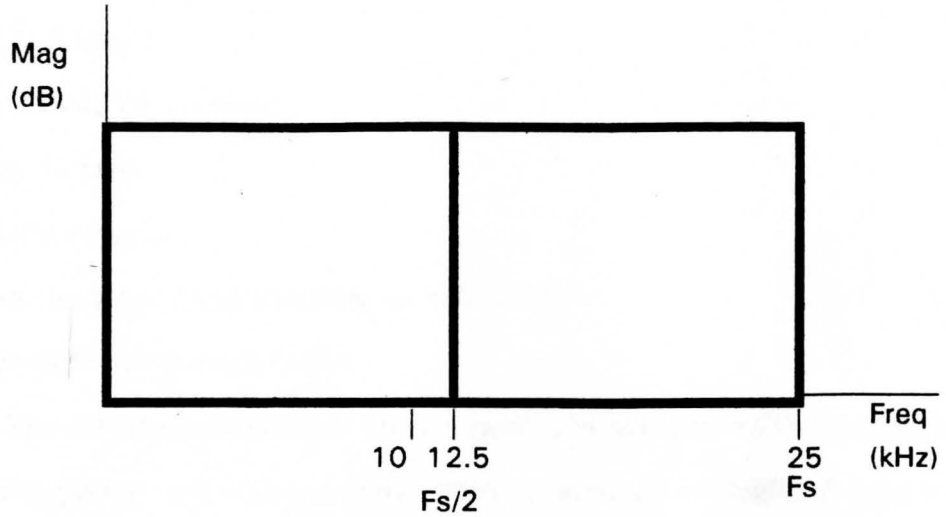



Figure 2. Transfer Function Without Low Pass Anti-Aliasing Filter

## System Clock And Timing

The software flow diagram of the AFA is shown in Figure 3. This flowchart contains the steps listed earlier to complete the program. The flowchart shows that the AFA will run in a continuous loop. The main objective is to extract digital data from the ADC, run five infinite impulse response (IIR) bandpass filter routines, detect the peak amplitude of each filter, encode the peak amplitude of each filter, encode the peak value to an LED value, and update the QSPI transmit registers, which outputs the information to the LED array.

The MC68HC16Z1 runs on a 16.78 MHz which has a system clock period of 60 nsecs. The processing of the signal should take no longer than 40080 nsecs which is about 668 clock cycles. This is shown below:

Fs=Sampling frequency

Ts=Sampling period

Fc=MC68HC16Z1 CPU clock frequency

Tc=MC68HC16Z1 CPU clock period

Fs=224.9 kHz

Ts=1/Fs=40.08 microsec

Fc=16.78 MHz

Tc=1/Fc=60nsec

System clock cycles per sampling period=Ts/Tc

Ts/Tc=668 system clock cycles

The sampling period is shown in Figure 2, which shows the relationship between sampling periods and real time digital signal processing. In DSP, all calculations and internal/external communications must be taken care of within the given sample period. Figure 4 relates the software flowchart in Figure 3 to the designated sample period.

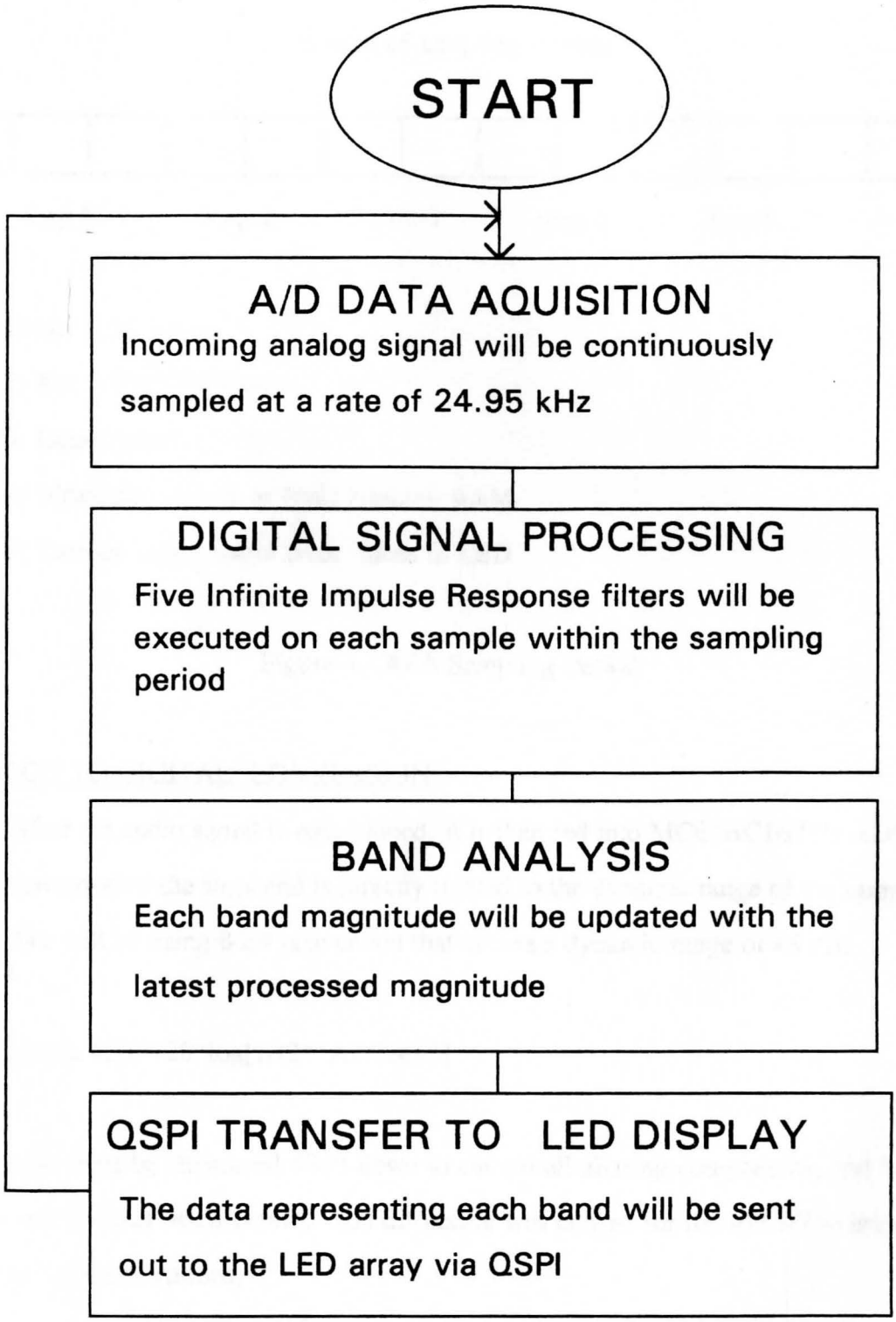
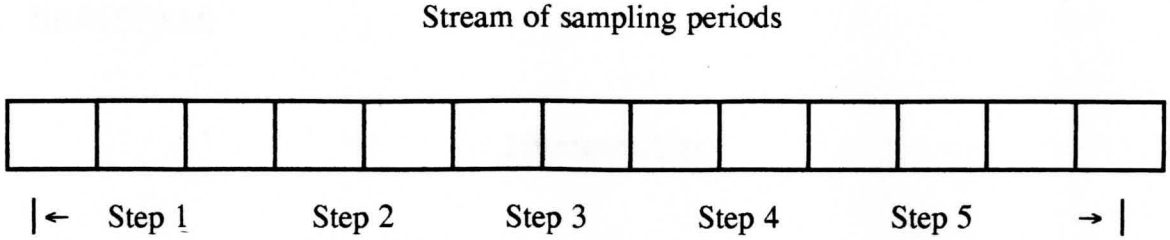



Figure 3. AFA Software Flow Diagram




Step 1: Get ADC value

Step 2: Run 5 IIR DSP routine

Step 3: Detect peak of each filter

Step 4: Write peak values to QSPI transmit RAM

Step 5: Turn on QSPI output peak values to LED

Figure 4. AFA Sampling Period

ANALOG TO DIGITAL CONVERSION

After the audio signal is conditioned, it is then fed into MC68HC16Z1's ADC. The attenuation of the stopband is directly related to the dynamic range of the sampled data. We will be using 8-bit conversion that allows a dynamic range of 48 dB.

Attenuation (dB)=20*log[1/(2**ADCres.)]

The signal must be attenuated 48dB down to cut out all aliasing components, and have a drop-off slope of 96dB/octave. 100 dB/octave was chosen for the roll-off to insure meeting the specifications.

# HARDWARE

## Microcontroller

The MC68HC16Z1 is a high speed 16-bit control unit designed and manufactured by Motorola. MC68HC16Z1 is a member of the M68300/68HC16 Family of modular microcontrollers. This microcontroller incorporates a true 16-bit CPU, a system integration module (SIM), an 8/10-bit analog to digital converter (ADC), a queued serial module (QSM), a general-purpose timer (GPT), and a 1024-byte standby RAM (SRAM). All these modules are interconnected by the intermodule bus (IMB). Although all these capabilities might be present in other microcontrollers, for the first time Motorola designers have included instructions and hardware necessary to implement control-oriented digital signal processing functions with a minimum of interfacing, in the MC68HC16Z1. A multiply and accumulate unit provides the capability to multiply signed 16-bit fractional numbers and store the resulting 32-bit fixed point product in a 36-bit accumulator. Modulo addressing supports finite impulse response filters; this feature was the main reason for selecting the MC68HC16Z1 for this design. The DSP techniques and all of the complex mathematical routines involved in DSP boil down to multiplication and addition. There are four internal registers, which facilitates DSP mathematical routines [2].

This microcontroller has an embedded module for ADC conversion. The ADC is a unipolar, successive-approximation converter with eight modes of operation. It has selectable 8 or 10-bit resolution.

The queued serial module (QSM) provides the MC68HC16Z1 with two serial communication interfaces divided into two submodules: the queued serial peripheral interface (QSPI) and the serial communications interface (SCI). The QSPI is a full-

duplex, synchronous serial interface for communicating with peripherals and other MCUs. It is enhanced by the addition of a queue for receiving and transmitting data.

The general-purpose timer (GPT), a module in MC68HC16Z1 is a 11-channel timer for use in systems where a moderate level of CPU control is required. The GPT is made up of different submodules: the compare/capture unit, the pulse accumulator, and the pulse-width modulation unit.

The standby RAM (SRAM) is a 1kbyte array of fast (two bus cycle) static RAM, which is especially useful for system stacks and variable storage.

The system clock for the MC68HC16Z1 is 16.78 MHz, and high-density complementary metal-oxide semiconductor (HCMOS) architecture makes the basic power consumption of MC68HC16Z1 low.

## Audio Frequency Analyzer

Spectral analysis is a method of determining the specific frequency content of a signal and the energy levels of these frequencies. The energy level, in this case the peak voltage, is processed by either Fourier Transform methods or by specific filtering of the signal. The result is then tabulated for more analysis or displayed in a visual format. This AFA is built by implementing digital signal processing filter techniques.

The input consists of two stereo audio signals that are added together with an op-amp summing circuit. The output is then sent to the anti-aliasing filter to remove unwanted high frequency components. The biasing circuit is used to adjust the signal to alter between $\pm 2.5$ volts for proper analog-to-digital conversion. The digital signal processing takes place in the MC68HC16Z1. MAX 274 manufactured by Maxim is used as a low frequency filter. MAX 274 is an 8th order, programmable, continuous-time active filter. There are four 2nd-order filter sections in MAX 274. MAX 274 uses a four-amplifier design which is insensitive to parasitic capacitances and high

bandwidth. The built-in capacitors and amplifiers, together with external resistors, form cascaded integrators with feedback to provide simultaneous lowpass and bandpass filtered outputs.

MC14489 is selected to drive 16 LEDs and three of them are cascaded to drive a total of 40 LEDs. MC14489 receives the signal on its serial input from QSPI of the MC68HC16Z1.

## SOFTWARE

A program has to be developed in order for Audio Frequency Analyzer to work properly. The main program is divided into 4 sections; each of these sections will perform a distinct operation. The first program is initializing the ADC submodule of MC68HC16Z1 and digitizes the analog input. The second program initializes the QSPI and defines the serial communication between the MC68HC16Z1 and MC14489. The third program activates the PIT and synchronizes the system operation with the incoming stream of data. The fourth and final program subjects the digitized data to five DSP routines to detect the peak of the samples at 125 Hz, 500 Hz, 1 kHz, 4 kHz, and 10 kHz (See Appendix B).

# CHAPTER II

# ADC CONVERSION

## SUCCESSIVE-APPROXIMATION ADC CONVERTER

### Overview

The successive-approximation method of converting an analog signal to a digital signal is used when the analog voltage changes rapidly and where the speed of conversion is a factor. In this method the counter is first reset, then the MSB is set to a 1. The DAC converter generates a voltage caused by the MSB, and this is compared to the input analog voltage. If the DAC voltage is greater than the input voltage, the MSB is reset. Then the second MSB is set to a 1, and the comparisons are made again. This process is repeated for each bit until the proper combination of bits is present.

The MC68HC16Z1 uses the successive-approximation method to convert the incoming audio signal.

## HARDWARE[1]

The ADC submodule is capable of performing 8-bit single conversion in 8 microseconds, and a 10-bit single conversion in 9 microseconds. The ADC functions are divided into three basic subsystems: an analog front end, a digital control section, and a bus interface. A block diagram of the converter is shown in Figure 5.

---

[1]For more detailed description of the ADC hardware see "The Technical Summary 16-Bit Modular Microcontroller".

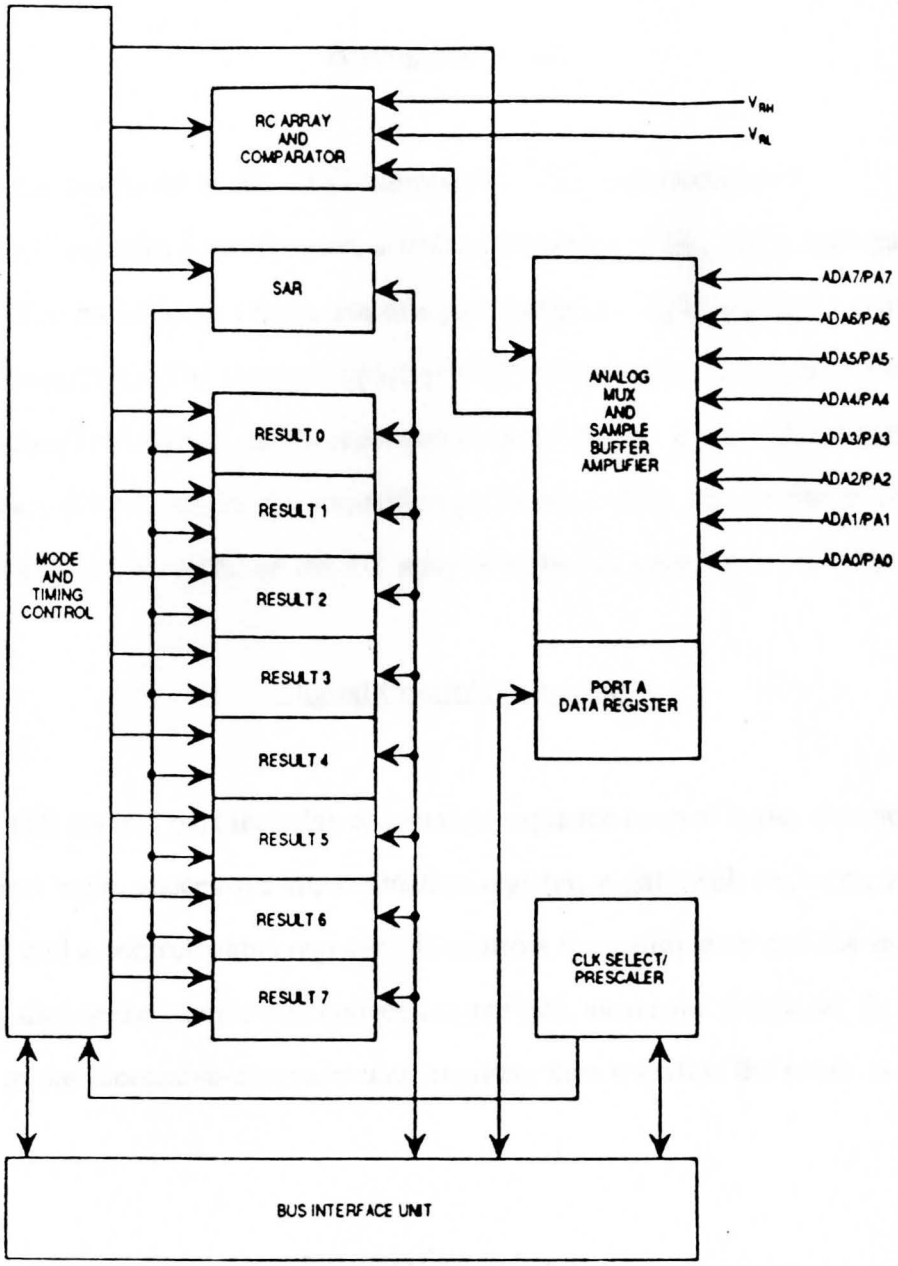



Figure 5. Analog-to-Digital Converter Block Diagram[2]

[2]This diagram shows the unconventional method of ADC which utilizes software for operation.

## Analog Subsystem

The analog front end of the ADC submodule of this microcontroller (MC68HC16Z1) includes a multiplexer, a resistor-capacitor array, and a high-gain comparator. The multiplexer selects one of eight internal or eight external signal sources for conversion. The resistor-capacitor (RC) array performs two functions, it acts as a sampled/hold circuit, and it accommodates the digital-to-analog comparison output necessary for successive approximation conversion. The comparator indicates whether each successive output of the RC array is higher or lower than the unit input.

## Digital Control Subsystem

The digital control part includes conversion sequence control logic, channel and reference select logic, successive approximation register, eight result registers, a port data register, and a control/status register. It controls the multiplexer and the output of the RC array during the sample and conversion periods, stores the results of comparison in the successive-approximation register, then transfers the result to a result register.

## Bus Interface Subsystem

The bus interface contains the logic circuitry which is necessary to interface the ADC to the intermodule bus. The ADC acts as a slave device on the bus. The interface must respond with the appropriate bus cycle termination signals and supply appropriate interface timing to the other submodules.

## SOFTWARE[3]

Four registers must be configured for ADC operation. These registers are the Module Configuration Register, Control Register 0, Control Register 1, and Status Register. In addition to these four registers there are two general-purpose registers which are used for any type of ADC conversion. These two registers are discussed first in the following sections.

### Successive Approximation Register (SAR):

The successive approximation register accumulates the result of each conversion one bit at the time, starting with the most significant bit. At the start of the resolution period, the MSB of the SAR is set, and all less significant bits are cleared. Depending on the result of the first comparison, the MSB is set or cleared. Each successive bit is set or left cleared in descending order until all eight or ten have been resolved. When conversion is complete, the content of the SAR is transferred to the appropriate result register.

### Result Registers

Result registers are used to store data after the conversion is complete. The registers can be accessed from the IMB under ABIU control. Each register can be read from three different addresses in the ADC memory map. The format of the result data

---

[3]There are more than one hundred memory mapped hardware registers that need to be configured for this project. I have chosen the ADC converter as an example to show how these registers and their related fields are selected and configured.

depends on the address from which it is read. Three types of formats can be specified, unsigned right-justified format {$FFF710:$FFF71E} , signed left-justified format {$FFF720:$FFF72E} , and unsigned left-justified format {$FFF730:$FFF73E} .

Module Configuration Register (ADCMCR):

The module configuration register contains five fields. This register selects the normal mode of operation for ADC and initializes the ADC operation. Bit 15 is set to 0 for normal operation. Bits 13 and 14 are cleared and bit 7 is also cleared to have unrestricted access to ADC submodule. The fields associated with this register are shown below:

| 15 | 14 | 13 | 12 ... 8 | 7 | 6 ... 0 |
|---|---|---|---|---|---|
| STOP | FRZ | | NOT USED | SUPV | NOT USED |
| 0 | 0 | 0 | | 0 | |

Figure 6. ADCMCR Configuration

ADC Control Register 0 (ADCTL0):

This register is used to define ADC clock source and to set up prescaling. Storing data in this register has an immediate effect. There are three fields that need to be defined. The Prescalar Rate Selection field is defined to select a divisor value which corresponds to minimum and maximum system clock. ADC clock is generated from system clock using a modulo counter and a divide-by-two circuit. The way the ADC

clock is selected is based on the value in the PRS region of ADCTL0. The system clock is divided by the PRS value plus one, then sent to the divide-by-two circuit. Bits 0 and 1 are set to select 16 MHz frequency for the ADC and the rest of the bits are cleared. The fields associated with this register are shown below:

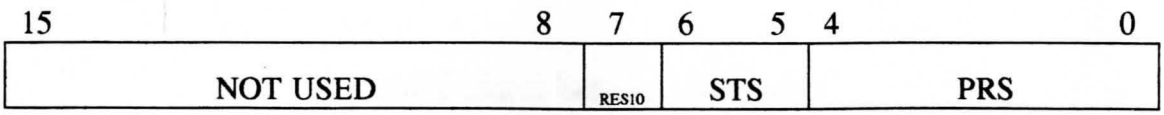



Figure 7. ADCTL0 Configuration

ADC Control Register 1 (ADCTL1):

ADCTL1 is used to start analog to digital conversion. It has four functions. The first one is to select a channel or a block of inputs for ADC conversion. Bits 0 thru 3 {CD:CA} are used to specify which input line or block of data is to be converted to digital. The audio signal in this project is applied to the input line AD0, so the binary value 0000 is picked for this region. The second function of this register is to select the kind of conversion. A single or a continuous conversion can be chosen. If the SCAN bit is set, a continuous conversion takes place and if this bit is cleared, a single conversion is done. The third selection is accomplished by setting or clearing the multichannel conversion bit (MULT). The fourth field in this register is a bit which, if it is zero, four-conversion sequence takes place and if it is one, sequential conversion of a block of four or eight channels (selected by channel selection field) is performed. The fields associated with this register are shown below:

| 15 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| NOT USED | SCAN | MULT | S8CM | CD | CC | CB | CA |

Figure 8. ADCTL1 Configuration

ADC Status Register (ADSTAT):

The last register that must be configured for proper operation of Analog Digital Converter submodule is its status register. This register is used to signal the CPU when the conversion is done. It has three active fields which will be discussed in detail here.

Conversion Complete Field (CCF):

Each bit {0:7} in this field corresponds to an ADC result register (CCF7 to RSLT7, etc.). A bit is set when conversion for the corresponding channel is complete, and remains set until the result register is read. It is cleared when the register is read.

Conversion Counter Field (CCTR):

This region shows the contents of the conversion counter pointer in either four or eight count conversion sequence. The value corresponds to the number of the next result register to be written, and thus indicates which channel is being converted.

Sequence Complete Flag (SCF):

SCF is set at the end of the conversion sequence when SCAN is cleared, and at the end of the first conversion sequence when SCAN is set. SCF is cleared when ADCTL1 is written and a new conversion sequence begins. The fields associated with this register are shown below:

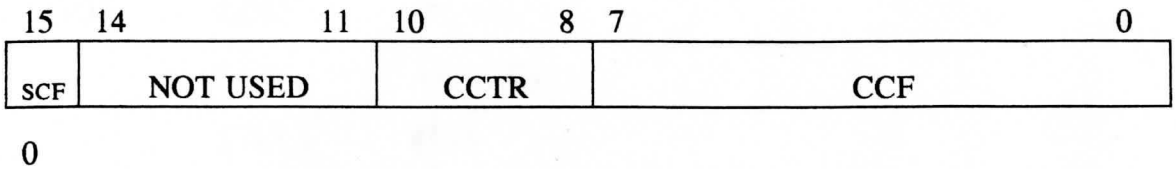



Figure 9. SCF Configuration

ADC PROGRAM

The program which initializes and begins converting analog input to digital signal is shown below. First, the ADMCR register is configured for normal operation. In some applications the analog input lines could be used as an I/O port. In this project the AD0 thru AD7 are selected as analog input lines. Second, in ADCTL0 register, bit 7 is set to zero for 8-bit conversion, and PRS field is filled with 00011 for a 16 MHz prescaled rate for clock generation. Third, ADCTL1 is filled with 0000 to ensure the single conversion sequence will run on a single channel (AD0). Fourth, ADSTAT is configured so that the result register 7 will hold the digital equivalent of the incoming analog signal. The complete list is shown on the next page.

```
*****     ADC Initialization     *****
          LDD       #$0000
          STD       ADCMCR
          LDD       #$0003
          STD       ADCTL0
*****     ADC Start     *****
LOOP      LDD       #$0000
          STD       ADCTL1
          LDAA      #$80
SCFSET    BITA      ADSTAT
          BEQ       SCFSET
          BRA       LOOP
```

# CHAPTER III

# DIGITAL SIGNAL PROCESSING

## INFINITE IMPULSE-RESPONSE (IIR) FILTER

The theory of the infinite impulse response (IIR) algorithm used to perform the bandpass filtering is examined and developed briefly. The equations presented here are developed to aid in the calculation of the coefficients necessary for digital signal processing (DSP). This algorithm is used to design five digital filters with different bandpass and cutoff frequencies. The following questions need to be answered. If the characteristics of a filter are defined in analog domain, how are those characteristics employed in digital domain? What is the connection between the analog passive filter and the digital IIR filter?

To outline the characteristics of the digital filter, the equivalent analog passive RCL bandpass network is considered. By applying the voltage divider rule, the transfer function can be written as follows:[3]

$$\frac{V_0}{V_i} = \frac{R}{R + j(\omega L - 1/\omega C)} \tag{1}$$

where $\omega = 2\pi f$. The gain is the magnitude of Equation (1),

$$A(\omega) = \frac{1}{\sqrt{[1 + Q^2(\frac{\omega^2 - \omega_0^2}{\omega\ \omega_0})^2]}} \tag{2}$$

where $\omega_0 = \frac{1}{\sqrt{LC}}$ and $Q = \frac{\omega_0 L}{R}$. The phase angle, $\phi$, is found by taking the ratio of the imaginary to real parts of the transfer function of Equation (1):

$$\phi = \tan^{-1}[Q\frac{(\omega^2 - \omega_0^2)}{(\omega\omega_0)}] \tag{3}$$

The equivalent s-plane expression is calculated by substituting $s = j\omega$:

$$H(s) = \frac{Rs}{Rs + Ls^2 + 1/C} \tag{4}$$

An op-amp active-filter circuit with essentially the same response as the passive RCL network is shown in Figure 10. This active filter has several advantages over the passive network: it eliminates the inductor; it is essentially isolated from input and output loading and can provide signal gain to the system.[4]

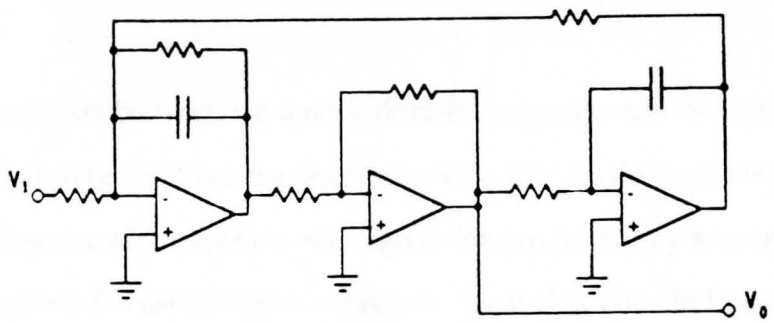



Figure 10. Op-Amp Active Bandpass Filter

A digital-transfer function representation of Equation (4) may be obtained by applying the bilinear transformation:[5,6]

$$s = \frac{2}{T}\frac{(1 - z^{-1})}{(1 + z^{-1})} \tag{5}$$

where $z = e^{j\theta}$, $\theta = \omega T$, and $T$ is the sample period. Equation (5) can also be expressed as follows:

$$\omega = \frac{2}{T} \tag{6}$$

using the definition of s and z. Substituting Equation (5) into Equation (4) yields the z-plane transfer function:

$$H(z) = \frac{\alpha(1 - z^{-2})}{1/2 - \gamma z^{-1} + \beta z^{-2}} \tag{7}$$

where the coefficients, $\alpha$, $\beta$, and $\gamma$, are related to R, C, and L by

$$\alpha = \frac{RT/2}{T^2/2C + RT + 2L} \tag{8}$$

$$\gamma = \frac{2L - T^2/2C}{T^2/2C + RT + 2L} \tag{9}$$

$$\beta = \frac{T^2/4C - RT/2 + L}{T^2/2C + RT + 2L} \tag{10}$$

The nonlinear relation between the analog domain frequency and the digital domain frequency is often referred to as the frequency warp.[5] As the frequency starts from zero, both analog domain frequency and digital domain frequency are approximately equal, since $\tan\theta \approx \theta$ for small angles. However, as analog domain frequency approaches infinity, digital domain frequency approaches $2\pi F_S/2$. In this particular application, most of the interesting and useful frequencies satisfy the small angle approximation (SAA) where $\theta < \pi/4$. Thus, using the SAA simplifies the digital analysis, and a direct correspondence to the analog RCL network is established. Although the SAA indeed simplifies the analysis, it must be used very carefully in the derivation because the nature of this IIR filter depends on very small differences of numbers. If not used correctly, the SAA can mask these differences and give totally erroneous results.

## THE DIFFERENCE EQUATION

To implement the transfer function from Equation (7) as an IIR filter, it is necessary to transform it to a difference equation in the discrete time domain. In this form, the filter can be directly implemented in software. Applying the inverse z-transform operator, $Z^{-1}$, to Equation (7) yields the following:[7]

$$Z^{-1}\{H(z)\} = Z^{-1}\{Y(z) / X(z)\} \tag{11}$$

$$Z^{-1}\{Y(z)[1 / 2 - \gamma z^{-1} + \beta z^{-2}]\} = Z^{-1}\{X(z)[\alpha(1 - z^{-2})]\} \tag{12}$$

The time delay property of the z-transform can be stated as follows:

$$X(z)z^{-m} = Z\{[x(n - m)]\} \tag{13}$$

where n is the discrete time index variable associated with continuous time sampling at a rate T. Evaluating Equation (6b), using the property of Equation (7), gives the final IIR difference equation:

$$y(n) = 2\{\alpha[x(n) - x(n - 2)] + \gamma y(n - 1) - \beta y(n - 2)\} \tag{14}$$

The coefficients, $\alpha$, $\beta$, and $\gamma$, in the difference equation (Equation (14)) are used to adjust the filter response (gain and phase as a function of frequency). The representation of the time-varying data is based on standard notation used in digital filter theory.[5,6,7] Thus, x(n) is the current sampled data represented as an N-bit signed fraction; x(n-1) is the previous data word; and x(n-2) is the data word previous to x(n-1). The time index is n and it is assumed that the sampled period, T, is constant and is related to the sample frequency, $f_S$, by $T=1/f_S$. For example, the time between

x(n) and x(n-2) is 2T. Sampled values of the input signal are only collected at integral multiples of T (i.e., the x(n)'s are standard sampled/digitized data).

The y(n) is similar to the input data, x(n), and represents the output data from the difference equation algorithm. As before, y(n) is the current output value; y(n-1) is the previous value; and y(n-2) is the value previous to y(n-1). Even though it is assumed that the input data is a signed fraction (a number between one and minus one), the y(n)'s can be greater than one (or less than minus one) unless scaling is performed to prevent this overflow condition.

The coefficients, $\alpha$, $\beta$, and $\gamma$ , in the difference equation (Equation (14)) are also fractional values (i.e. between one and minus one). As it will later be shown, scaling at the output can be controlled by imposing the following condition on two of these these coefficients:

$$\alpha = 1/4 - \beta/2 \qquad (15)$$

This formula guarantees that, at the center frequency, the gain is one and the phase difference is zero. In this case, the bandpass filter acts as an attenuation filter and a phase shifter for all frequencies other than the center frequency. Limiting the gain to one and the input to a fraction scaled to a maximum of one does not always prevent overflow at the output .

## RESPONSE OF THE DIGITAL FILTER

The gain and phase response can be calculated solely from Equation (7). The advantage of complex numbers is that both gain and phase information are present in the transfer function. By definition, the gain is the absolute magnitude of H(z). In the RLC circuit , the gain is simply the ratio of the resistance to the magnitude of the total

complex impedance. The ratio of the real to imaginary components of the impedance is equal to the tangent of the phase. Likewise, the ratio of real to imaginary components of H(z) is equal to the tangent of the phase for the digital case.

Euler's identity is implemented to ease the calculation of gain, $G(\omega)$, and phase, $\theta(\omega)$:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{16}$$

The transfer function (Equation (7)) then becomes:

$$H(e^{j\theta}) = \frac{\alpha(e^{j2\theta} - 1)}{0.5e^{j2\theta} - \gamma e^{j\theta} + \beta} \tag{17}$$

and

$$H(e^{j\theta}) = \frac{\alpha(\cos 2\theta - 1) + j\sin 2\theta}{(0.5\cos 2\theta - \gamma\cos\theta + \beta) + j(0.5\sin 2\theta - \gamma\sin\theta)} \tag{18}$$

where $\theta = \omega T$. For example, $\theta = \pi/2$ would correspond to $f = f_S/4$ (since $\omega = 2\pi f$ and $T = 1/f_S$). If $f_S = 44.1$ kHz, then $\theta = \pi/2$ would be a frequency of 11.025 kHz.
The filter gain is found by evaluating the following expression:

$$G(\omega) = \sqrt{H(e^{j\theta})H^*(e^{j\theta})} \tag{19}$$

and, after some algebraic and trigonometric manipulations, becomes:

$$G(\omega) = \frac{2\alpha\sin\theta}{\{[(1/2 - \beta)\sin\theta]^2 + [(1/2 + \beta)(\cos\theta - \cos\theta_0)]^2\}^{1/2}} \tag{20}$$

where

$$\cos\theta_0 = \gamma(1/2 + \beta) \tag{21}$$

is the filter center frequency.

Examination of the gain in Equation (13) shows several important features:

- The gain, $G(\omega)$, is proportional to $\alpha$.
- The gain at the center frequency, $\omega_0$, is

$$G_0 = 2\,\alpha/(1/2 - \beta) \tag{22}$$

as previously noted in Equation (9).

- The bandwidth is adjusted by $\beta$ (that also affects the center frequency as shown by Equation (21)).
- Equation (20) is symmetric (neglecting the zero at $\theta=\pi$) on a logarithmic scale. This characteristic of the gain can be seen more easily by taking the SAA of Equation (20) where

$$\sin\theta \approx \theta \tag{23}$$

$$\cos\theta \approx 1 - \theta^2/2 \tag{24}$$

so that

$$G_a(\omega) = \frac{G_0}{\sqrt{1 + \left(\frac{0.5 + \beta}{0.5 - \beta}\right)^2 \left(\frac{\theta_0^2 - \theta^2}{2\theta}\right)^2}} \tag{25}$$

Substitution of $\theta = k\theta_0$ or $\theta = \theta_0/k$ yields equivalent values of gain, thus proving that the gain is symmetrical over the log of frequency. The subscript "a" denotes that the SAA was used in that expression.

The phase shift, $\varphi(\omega)$, is found from the ratio of the imaginary to real part of H(z) from Equation (17):

$$\tan\phi = \frac{\mathrm{Im}[H(e^{j\theta})]}{\mathrm{Re}[H(e^{j\theta})]} \tag{26}$$

After some algebraic and trigonometric manipulations, Equation (26) can be written as

$$\tan\phi = \frac{(0.5+\beta)(\cos\theta - \cos\theta_0)}{(0.5-\beta)\sin\theta} \tag{27}$$

Applying the SAA simplifies the previous result:

$$\tan\phi_a = \frac{(0.5+\beta)(\theta_0^2 - \theta^2)}{(0.5-\beta)2\theta} \tag{28}$$

The SAA can be used to approximate the filter center frequency, $\theta_0$, from Equation (21):

$$\theta_{0a} = \sqrt{\frac{1+2\beta-2\gamma}{0.5+\beta}} \tag{29}$$

The SAA is accurate within a few percent for angles up to $\pi/4$. (This SAA corresponds to a filter frequency of $f < f_S/8$.)

The bandwidth of the filter is most easily determined from Equation (25). Generally, two frequencies are considered, one on each side of the center frequency, $\theta_0$. The gain at each of the frequencies, $\theta_1$ and $\theta_2$, is equivalent and is commonly chosen so that the value of gain is $G_0/\sqrt{2}$ =-3 dB of the center frequency gain. As previously noted, $\theta_1 = \theta_0/k$ and $\theta_2 = k\theta_0$ for a filter symmetric about the center frequency over the log of frequency.

The Q of the filter in such a case is as follows:

$$Q = \frac{\theta_0}{\Delta\theta} = \frac{\theta_0}{k\theta_0 - \theta_0/k} = \frac{k}{k^2-1} \tag{30}$$

where $k > 1$. Since, by definition, the bandwidth is determined at the frequencies corresponding to a gain of $G_0/\sqrt{2}$, using Equation (25), the following term is equal to one:

$$[(0.5+\beta_a)/(0.5+\beta_a)]^2[(\theta_1^2-\theta_1^2)/2\theta_1]^2=1 \tag{31}$$

and using Equation (30) to solve for $\beta_a$ in terms of Q yields

$$\frac{0.5+\beta_a}{0.5-\beta_a}=\frac{2}{\theta_0}\bullet\frac{k}{k^2-1}=\frac{2Q}{\theta_0} \tag{32}$$

Rearranging terms results in the final form:

$$\beta_a=\frac{Q-\theta_0/2}{2Q+\theta_0} \tag{33}$$

where $\theta_0=2\pi f_0/f_S$. The subscript "a" is used to denote that the SAA was used in this derivation (i.e., by definition of Q from Equation (25)).

Solving for $\gamma$ in Equation (21) gives

$$\gamma=(0.5+\beta)\cos\theta_0 \tag{34}$$

For unity gain at the center frequency, $\alpha$ (Equation (22)) becomes

$$\alpha=(0.5-\beta)/2 \tag{35}$$

Equation (25) now simplifies to the following equation for gain:

$$G_a(\omega)=\frac{1}{\sqrt{1+Q^2[(\theta_0^2-\theta^2)/(\theta_0\theta)]^2}} \tag{36}$$

Equation (28) becomes

$$\phi_a(\omega)=\tan^{-1}\{Q[(\theta_0^2-\theta^2)/(\theta_0\theta)]\} \tag{37}$$

Equations (20), (21), (22), and (27) provide a complete, theoretically accurate, and concise description of the digital filter response described by the difference equation (Equation (14)). The coefficients, $\alpha$ and $\gamma$, can be found from $\beta$, $\theta_0$, and $G_0$. $\beta$ must be determined from the gain (Equation (20)) by picking $G(\theta_1)$ and $\theta_1$, then finding the values of $\beta$ from the equality. These four equations are exact and can be used over the entire frequency range from 0 to $\pi$.

Equations (33) through (37) provide a simplified set of formulas that are reasonably accurate for $f<f_S/8$ and for unity gain at the center frequency.

## ANALYSIS CONSTRAINTS OF THE BILINEAR TRANSFORMATION

The passive series resonant network shows how to determine the IIR coefficients from the RCL values of a passive network filter based on the bilinear transformation. This technique is very powerful, especially if the frequencies of interest are much lower than the sample frequency (as is often the case in digital audio applications) so that the SAA can be used. The resonant and cutoff frequency and quality factor, Q, of most RCL networks are known or can be easily determined. The difference equation shows how to convert the transfer function to another difference equation, which is the final form (for software implementation) of the digital IIR filter. The relationship connecting the R, L, and C values of the analog filter to the coefficients $\alpha$, $\beta$, and $\gamma$ of the digital filter (from Equations 8 thru 10) holds true only for frequencies where the SAA is valid. This frequency range makes up the linear region of the bilinear transformation where (from Equation (6)) $\tan \theta \approx \theta$. In this case, a direct correspondence between the response of almost any RCL network and an IIR filter's coefficients can be established, as previously demonstrated for the bandpass filter network. This technique lends itself to audio applications because the response of a network is usually described as a function of the log of frequency. The audio range is basically logarithmic, thus the SAA applies to most of the range of interest since $f_S/8$ is very close to $f_S/2$ on a log scale.

## COEFFICIENT QUANTIZATION

Coefficient quantization depends solely on the word length of the filter coefficients. Equations (20), (21), and (22) yield values for $\alpha$, $\beta$, and $\gamma$ for given values of center frequency and bandwidth. These formulas are exact. However, the

word size of the variables used to represent the coefficients in the filter algorithm are of finite length. Therefore, only certain discrete values of center frequency, bandwidth, and resonant frequency gain are obtainable.

To analyze the effects of coefficient quantization in this particular digital filter, let N be the number of bits used to represent data in the algorithm. Assuming that the coefficients are fractions, the smallest number that can be represented is therefore:

$$\delta = 2^{-(N-1)} \tag{38}$$

Using Equation (38), $\alpha$, $\beta$, and $\gamma$ can be represented as follows:

$$\gamma = 1 - n\delta \tag{39}$$

$$\beta = 1/2 - m\delta \tag{40}$$

since $\beta < 1/2$ and $|\gamma K|$. This can be easily seen by evaluating the zeros of the transfer function (Equation 7) and then calculating the magnitude of that complex number. The resulting value is the distance from the origin to the pole in the complex plane and is equal to $2\beta$. Now, since the poles must lie within the unit circle $\beta < 1/2$.[7] Using Equation (19), it can be seen that $|\gamma| < 1$. The integers n and m take on values from 1 to $2^{N-1}$. Equation (29) can be written as

$$\theta_0 = \sqrt{\frac{1 + 2(1/2 - m\delta) - 2(1 - n\delta)}{1/2 + (1/2 - m\delta)}} = \sqrt{\frac{2(n - m)\delta}{1 - m\delta}} \tag{41}$$

By inspection, the lowest nonzero value of $\theta_0$ is with n=2 and m=1. The lowest obtainable frequency is then

$$f_0 = \theta_0 f_s / 2\pi = \frac{1}{2\pi}\sqrt{2\left(\frac{\delta}{1 - \delta}\right)} = f_s 2^{-N/2/\pi} \tag{42}$$

Assuming $f_S$=44.1 kHz. The lowest obtainable frequency for 16 bits is 54.8 Hz; for 24 bits, it is 3.4 Hz. Clearly, 16 bits does not yield the coefficient accuracy needed to implement filter responses in the low-frequency bands (i.e., 20 to 200 Hz) for audio applications.

The sampling frequency in this design is 24.95 kHz, and the center frequency and Q for all five bandpass filters are:

| | $F_0$ | $F_s$ | Q |
|---|---|---|---|
| FIRST FILTER | 125 Hz | 24.95 kHz | 0.5 |
| SECOND FILTER | 500 Hz | 24.95 kHz | 1.0 |
| THIRD FILTER | 1 kHz | 24.95 kHz | 1.5 |
| FOURTH FILTER | 4 kHz | 24.95 kHz | 1 |
| FIFTH FILTER | 10 kHz | 24.95 kHz | 0.5 |

Using above values and Equations (34), (35), and (36) the coefficients can be calculated (See appendix A). After the coefficients are calculated, these fractions are converted to 16-bit hexadecimal values. Two's complement arithmetic is utilized to the calculation, therefore the most significant bit is the sign bit. The fraction is contained in the remaining bits, 0-14. A total of fifteen bits will represent the numbers from 0 to 32,767. The fraction first is multiplied by 32,767, and then is converted to its hexadecimal equivalent. If this number is negative, the two's complement is taken to find the final answer.

A series of multiplications and additions are performed to implement these values with the 68HC16. The following figure shows the memory configuration used to carry out the mathematics. As an example, the 1 kHz filter's coefficients are used.

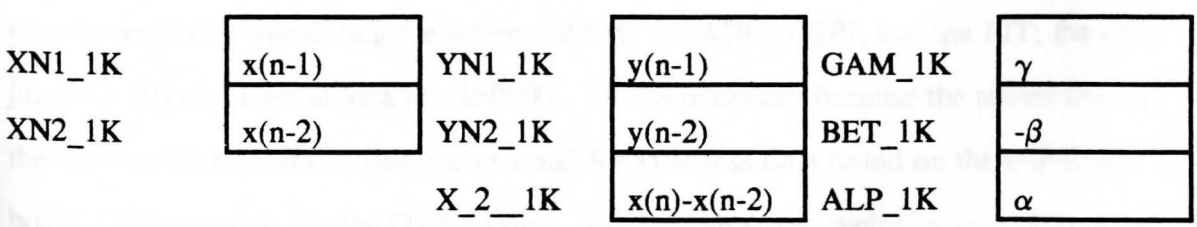



Figure 11. DSP Memory Configuration

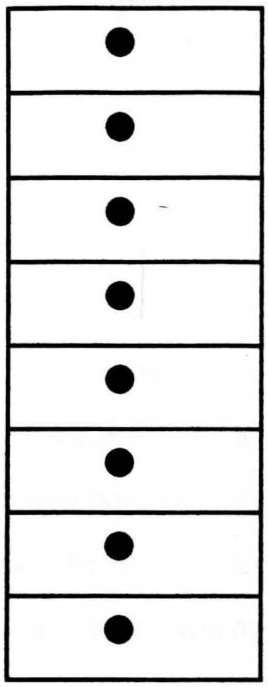
The ADC value x(n) is divided by two at the start of the algorithm. This is to ensure that no overflow will occur with the binary mathematics. Next, x(n-2) is subtracted from x(n). This value is stored away to location X_2_1K. Three MAC (multiply and accumulate) instructions starting at this address are executed. Then y(n-1) is multiplied by $\gamma$ and is added to the accumulator. The equation that defines $\beta$ above gives its positive value; this value is converted to its negative before storing it in memory. The next step is to multiply y(n-2) by $-\beta$ and add it to M accumulator and multiply [x(n)-x(n-2)] by $\alpha$. This too is added to the accumulator. The last operation is the multiplication of the accumulator by two. This is done by a left shift instruction. This gives the y(n) value. Now the x and y terms need to be updated before the next sample is received. In other words, x(n-1) becomes x(n-2) and x(n) becomes x(n-1). Also, y(n-1) becomes y(n-2) and y(n) becomes y(n-1). Once the DSP is finished on the input x(n) sample, the peak detect algorithm is executed.

## SIGNAL PEAK DETECTOR

The audio signal is digitally sampled at 24.95 kHz. The amplitude will be detected from the sampled signal and encoded with a value that represents its peak value according to a bar of eight LEDs. Using a reference value of 0.775 Vrms equivalent to 0dB, the signal is measured and displayed from -15dB to +6dB. Figure 13 illustrates the relationship between the LED bar, decibels, Vrms, and Vpeak. This process begins by initializing the internal RAM, the ADC, QSPI, and the PIT; the code jumps to internal Ram at location $F0000. This is necessary because the access time of the internal RAM is faster than the external RAM access time found on the evaluation board. This speed is for the DSP routine. The program then cycles in an infinite loop reading the ADC, encoding the ADC value to its equivalent LED value, and checking to see if the current value is greater than the previous peak value. If so, then the

current peak value is updated and stored away in memory. The code will increase the current peak value, not decrease it. It also updates the LED array every 10.26 msec with the current peak value. This routine uses a programmable interrupt timer (PIT) to detect the changes in incoming signal. The PIT times out every 62.5 msec to decrement the peak value and the LED display. If the input signal is sharply attenuated, the LED bar slowly decrements. After downloading the program, the 68HC16 is ready to analyze the audio signals.

A signal generator, voltmeter, counter , and an oscilloscope show the analyzer works precisely. A known signal sweeps and the results are displayed on the bar of LED's. The LED's are on and off, based on the magnitude of the input signal. The test verifies each filter is working properly and the software is reliable.



| LED BAR | db | Vrms | Vpeak |
|---|---|---|---|
| ● | +6 | 1.548 | 2.187 |
| ● | +3 | 1.096 | 1.548 |
| ● | 0 | 0.775 | 1.096 |
| ● | -3 | 0.549 | 0.775 |
| ● | -6 | 0.389 | 0.549 |
| ● | -9 | 0.275 | 0.389 |
| ● | -12 | 0.195 | 0.275 |
| ● | -15 | 0.138 | 0.195 |

$$dB = 20 * \log(V_{in}/V_{ref})$$

$$V_{ref} = 0.775\ V_{rms}$$

$$V_{peak} = 1.414 * V_{rms}$$

Figure 12. Relationship Between Signal Amplitude and The LED Bar

# CHAPTER IV

## CONCLUSIONS

In this thesis an introduction to the basic operation of the MC68HC16 microcontroller was presented. A program for a five band audio analyzer that used this microcontroller was written. An audio signal was converted to digital, and subjected to the five digital filters via software. Then the magnitude of each center frequency was detected and displayed on an array of LED's.

The project demonstrated the feasibility of the software design. A microcontroller could be used to digitally sample a signal within the audio range and to subject it to digital signal processing. The filtering of such a signal was accomplished by means of software. The programmability of the DSP made it much more flexible than analog approaches to the filter design. Instead of using conventional method of filtering which uses components such as resisters, capacitors, and operational amplifiers, the incoming signal was digitally (and by means of software) sampled, filtered, and then the strength of each band was serially transferred and displayed on an array of LEDs. The performance was verified by sweeping a known signal and observing the results on the bar of LEDs. The implementation of DSP was fully accomplished

The project also effectively demonstrates how the correct micro-controller with proper support circuitry can greatly simplify a design. Where a variety of interfacing hardware is integrated into one chip, the estimated cost, space and the speed are greatly improved. This integration of chips into one and the programmability of the microcontroller clearly demonstrate the advantage of modern digital techniques over the

analog method. The same microcontroller can be used to filter out other signals by making minor changes to the program without any hardware modification.

## AREAS FOR FURTHER DEVELOPMENT

In some applications it is required to design filters that have center frequencies higher than 10 kHz. In order to accomplish that, a program which takes fewer cycles to be executed must be developed. At the same time the program must maintain the routines that initialize the submodules and run digital signal processing instructions.

# APPENDIX A

## CALCULATION OF THE COEFFICIENTS

| | FIRST BAND | FRACTIONS | MULTIPLIED BY 32768 | HEX NUMBER |
|---|---|---|---|---|
| F0 | 1.25E+02 | 125.00000 | 4096000 | |
| FS | 2.50E+04 | 24950.00000 | 817561600 | |
| Q | 5.00E-01 | 0.50000 | 16384 | |
| TETA | 3.15E-02 | 0.03148 | 1031.500081 | |
| X | 3.15E-02 | 0.03148 | 1031.500081 | |
| TAN(X/2Q) | 7.87E-03 | 0.00787 | 257.880344 | |
| BETA | 4.92E-01 | 0.49219 | 16128.12806 | 3F32 |
| GAMA | 3.92E-01 | 0.99170 | 32496.02092 | 7F28 |
| ALFA | 3.90E-03 | 0.00390 | 127.935972 | 66 |
| | | IF X EXCEEDS 7.85E-01, THEN IT IS EQUAL TO 0.75398 | | |

| | SECOND BAND | FRACTIONS | MULTIPLIED BY 32768 | HEX NUMBER |
|---|---|---|---|---|
| F0 | 5.00E+02 | 500.00000 | 16384000 | |
| FS | 2.50E+04 | 24950.00000 | 817561600 | |
| Q | 1.00E+00 | 1.00000 | 32768 | |
| TETA | 1.26E-01 | 0.12592 | 4126.000324 | |
| X | 6.30E-02 | 0.06296 | 2063.000162 | |
| TAN(X/2Q) | 3.15E-02 | 0.03149 | 1031.840927 | |
| BETA | 4.69E-01 | 0.46946 | 15383.33843 | 3E68 |
| GAMA | 9.62E-01 | 0.96179 | 31515.83981 | 7E3F |
| ALFA | 1.53E-02 | 0.01527 | 500.3307872 | CB |
| | | IF X EXCEEDS 7.85E-01, THEN IT IS EQUAL TO 0.75398 | | |

| | THIRD BAND | FRACTIONS | MULTIPLIED BY 32768 | HEX NUMBER |
|---|---|---|---|---|
| F0 | 1.00E+03 | 1.00E+03 | 32768000 | |
| FS | 2.50E+04 | 2.50E+04 | 817561600 | |
| Q | 1.50E+00 | 1.50E+00 | 49152 | |
| TETA | 2.52E-01 | 2.52E-01 | 8252.000647 | |
| X | 8.39E-02 | 8.39E-02 | 2750.666882 | |
| TAN(X/2Q) | 6.30E-02 | 6.30E-02 | 2065.730181 | |
| BETA | 4.41E-01 | 4.41E-01 | 14438.34771 | 3866 |
| GAMA | 9.11E-01 | 9.11E-01 | 29850.14251 | 749A |
| ALFA | 2.97E-02 | 2.97E-02 | 972.8261473 | 3CC |
| | | IF X EXCEEDS 7.85E-01, THEN IT IS EQUAL TO 0.75398 | | |

| | FOURTH BAND | FRACTIONS | MULTIPLIED BY 32768 | HEX NUMBER |
|---|---|---|---|---|
| F0 | 4.00E+03 | 4.00E+03 | 131072000 | |
| FS | 2.50E+04 | 2.50E+04 | 817561600 | |
| Q | 1.00E+00 | 1.00000 | 32768 | |
| TETA | 1.01E+00 | 1.00732 | 33008.00259 | |
| X | 5.04E-01 | 0.50366 | 16504.00129 | |
| TAN(X/2Q) | 2.57E-01 | 0.25729 | 8430.986702 | |
| BETA | 2.92E-01 | 0.29169 | 9558.001333 | 2556 |
| GAMA | 4.23E-01 | 0.42286 | 13856.26328 | 3620 |
| ALFA | 1.04E-01 | 0.10416 | 3412.999333 | D54 |
| | | IF X EXCEEDS 7.85E-01, THEN IT IS EQUAL TO 0.75398 | | |

| - | FIFTH BAND | FRACTIONS | MULTIPLIED BY 32768 | HEX NUMBER |
|---|---|---|---|---|
| F0 | 1.00E+04 | 1.00E+04 | 327680000 | |
| FS | 2.50E+04 | 2.50E+04 | 817561600 | |
| Q | 5.00E-01 | 0.50000 | 16384 | |
| TETA | 2.52E+00 | 2.51831 | 82520.00647 | |
| X | 7.54E-01 | 0.75398 | 24706.41664 | |
| TAN(X/2Q) | 1.91E-01 | 0.19076 | 6250.811315 | |
| BETA | 3.38E-01 | 0.33815 | 11080.41145 | 2B48 |
| GAMA | -6.81E-01 | -0.68055 | -22300.20051 | 2's Complement A8E4 |
| ALFA | 8.09E-02 | 0.08093 | 2651.794273 | A5B |
| | | IF X EXCEEDS 7.85E-01, THEN IT IS EQUAL TO 0.75398 | | |

# APPENDIX B

## COMPLETE LIST OF THE PROGRAM

```
*Description :   This program executes a band pass DSP filter
*                algorithm in real time.
*                The digital data is acquired with the on-chip
*                analog to digital converter.
*                The sampling frequency is 25 kHz.
*                Executes the function
*                y(n) = 2{α[x(n)-x(n-2)] + γy(n-1) - βy(n-2)}
*                The output of the 5 filters are analyzed for
*                their peak values and then encoded with a value
*                to represent each peak on the LED bar array.
*                These values are sent out to the LEDs via the
*                QSPI.
*************************************************************************

          INCLUDE 'EQUATES.ASM'      ;table of EQUates for
                                     ;common register addresses
          INCLUDE 'ORG00000.ASM'     ;initialize reset vector

***       Addresses of coefficients for the IIR Filters and initialization

COEFB     EQU     $0280            ;base addr of coefficients
GAM_125   EQU     COEFBS+$0        ;addr of the gamma coef
BET_125   EQU     COEFBS+$2        ;addr of the beta coef
```

```
ALP_125     EQU     COEFBS+$4       ;addr of the alpha coef
GAM_500     EQU     COEFBS+$6       ;addr of the gamma coef
BET_500     EQU     COEFBS+$8       ;addr of the beta coef
ALP_500     EQU     COEFBS+$A       ;addr of the alpha coef
GAM_1K      EQU     COEFBS+$C       ;addr of the gamma coef
BET_1K      EQU     COEFBS+$E       ;addr of the beta coef
ALP_1K      EQU     COEFBS+$10      ;addr of the alpha coef
GAM_4K      EQU     COEFBS+$12      ;addr of the gamma coef
BET_4K      EQU     COEFBS+$14      ;addr of the beta coef
ALP_4K      EQU     COEFBS+$16      ;addr of the alpha coef
GAM_10K     EQU     COEFBS+$18      ;addr of the gamma coef
BET_10K     EQU     COEFBS+$1A      ;addr of the beta coef
ALP_10K     EQU     COEFBS+$1C      ;ddr of the alpha coef
            ORG     $F0280
            dc.w    $7F28           ;125 Hz gamma coef, Q=0.5
            dc.w    $3F32           ;125 Hz beta coef, Q=0.5
            dc.w    $66             ;125 Hz alpha coef, Q=0.5
            dc.w    $7E3F           ;500 Hz gamma coef, Q=1.0
            dc.w    $3E68           ;500 Hz beta coef, Q=1.0
            dc.w    $CB             ;500 Hz alpha coef, Q=1.0
            dc.w    $749A           ;1k Hz gamma coef, Q=1.5
            dc.w    $3866           ;1k Hz beta coef, Q=1.5
            dc.w    $3CC            ;1k Hz alpha coef, Q=1.5
            dc.w    $3620           ;4k Hz gamma coef, Q=1.0
            dc.w    $2556           ;4k Hz beta coef, Q=1.0
            dc.w    $D54            ;4k Hz alpha coef,Q=1.0
            dc.w    $A8E4           ;10k Hz gamma coef, Q=0.5
```

```
        dc.w    $2B48           ;10k Hz beta coef, Q=0.5
        dc.w    $A5B            ;10k Hz alpha coef, Q=0.5

*****   Addresses of filter terms for the x(n) terms and initialization

XTRMBS  EQU     $02A0           ;base addr of x(n) filter terms
XN1_125 EQU     XTRMBS+$0       ;x(n-1)
XN2_125 EQU     XTRMBS+$2       ;x(n-2)
XN1_500 EQU     XTRMBS+$4       ;x(n-1)
XN2_500 EQU     XTRMBS+$6       ;x(n-2)
XN1_1K  EQU     XTRMBS+$8       ;x(n-1)
XN2_1K  EQU     XTRMBS+$A       ;x(n-2)
XN1_4K  EQU     XTRMBS+$C       ;x(n-1)
XN2_4K  EQU     XTRMBS+$E       ;x(n-2)
XN1_10K EQU     XTRMBS+$10      ;x(n-1)
XN2_10K EQU     XTRMBS+$12      ;x(n-2)
        ORG     $F02A0
        dc.w    $0000           ;125 Hz x(n-1)
        dc.w    $0000           ;125 Hz x(n-2)
        dc.w    $0000           ;500 Hz x(n-1)
        dc.w    $0000           ;500 Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)
        dc.w    $0000           ;1k Hz x(n-1)
        dc.w    $0000           ;1k Hz x(n-2)
```

```
*****       Addresses of filter terms for the y(n) terms and initialization

YTRMBS      EQU     $02C0           ;base addr of y(n) filter terms
YN1_125     EQU     YTRMBS+$0       ;y(n-1)
YN2_125     EQU     YTRMBS+$2       ;y(n-2)
X_2_125     EQU     YTRMBS+$4       ;x(n) - x(n-2), stored here for mac
YN1_500     EQU     YTRMBS+$6       ;y(n-1)
YN2_500     EQU     YTRMBS+$8       ;y(n-2)
X_2_500     EQU     YTRMBS+$A       ;x(n) - x(n-2), stored here for mac
YN1_1K      EQU     YTRMBS+$C       ;y(n-1)
YN2_1K      EQU     YTRMBS+$E       ;y(n-2)
X_2_1K      EQU     YTRMBS+$10      ;x(n) - x(n-2), stored here for mac
YN1_4K      EQU     YTRMBS+$12      ;y(n-1)
YN2_4K      EQU     YTRMBS+$14      ;y(n-2)
X_2_4K      EQU     YTRMBS+$16      ;x(n) - x(n-2), stored here for mac
YN1_10K     EQU     YTRMBS+$18      ;y(n-1)
YN2_10K     EQU     YTRMBS+$1A      ;y(n-2)
X_2_10K     EQU     YTRMBS+$1C      ;x(n) - x(n-2), stored here for mac
            ORG     $F02C0
            dc.w    $0000           ;125 Hz y(n-1)
            dc.w    $0000           ;125 Hz y(n-2)
            dc.w    $0000           ;125 Hz [ x(n) - x(n-2) ]
            dc.w    $0000           ;500 Hz y(n-1)
            dc.w    $0000           ;500 Hz y(n-2)
            dc.w    $0000           ;500 Hz [ x(n) - x(n-2) ]
            dc.w    $0000           ;1k Hz y(n-1)
```

```
        dc.w    $0000       ;1k Hz y(n-2)
        dc.w    $0000       ;1k Hz [ x(n) - x(n-2) ]
        dc.w    $0000       ;4k Hz y(n-1)
        dc.w    $0000       ;4k Hz y(n-2)
        dc.w    $0000       ;4k Hz [ x(n) - x(n-2) ]
        dc.w    $0000       ;10k Hz y(n-1)
        dc.w    $0000       ;10k Hz y(n-2)
        dc.w    $0000       ;10k Hz [ x(n) - x(n-2) ]

*****   Addresses of various temporary variables and initialization

PKRES   EQU     $02E0       ;base addr of filter result storage
PK_125  EQU     PKRES+$0    ;peak value for 125 Hz
PK_500  EQU     PKRES+$1    ;peak value for 500 Hz
PK_1K   EQU     PKRES+$2    ;peak value for 1k Hz
PK_4K   EQU     PKRES+$3    ;peak value for 4k Hz
PK_10K  EQU     PKRES+$4    ;peak value for 10k Hz
CNT     EQU     PKRES+$6    ;count value for LED qspi update ;
                            ;routine
AD      EQU     PKRES+$8    ;divided by two adc reading
        ORG     $F02E0
        dc.w    $0000       ;125 peak value, 500 peak value
        dc.w    $0000       ;1k peak value, 4k peak value
        dc.w    $0000       ;10k peak value
        dc.w    $0000       ;update count value
        dc.w    $0000       ;divided by two adc reading
        ORG     $0200
```

```
*****       Initialization Routines *****

            INCLUDE 'INITSYS.ASM'       ;initially set EK=F, XK=0,
                                        ;YK=0,ZK=0
                                        ;set sys clock at 16.78 MHz, disable
                                        ;COP

*****  RAM and Stack Initialization     *

            LDD     #$00FF
            STD     RAMBAH              ;store high ram array, bank F
            LDD     #$0000
            STD     RAMBAL              ;store low ram array, 0000
            CLR     RAMMCR              ;enable ram
            LDAB    #$0F
            TBSK                        ;set SK to bank F for system stack
            LDS     #$02FE              ;put SP in 1k internal SRAM

            INCLUDE 'SERIAL.ASM'

*****       Initialize level 6 autovector address

            LDAB    #$00
            TBEK                        ;ek extension pointer = bank0
            LDD     #INT_RT             ;load Dacc with interrupt vector
                                        ;addr
```

```
        STD     $002C           ;store addr to level 6 autovector

***** Initialize the PIT *****

        LDAB    #$0F
        TBEK                    ;ek extension pointer = bankf
        LDD     #$0610
        STD     PICR            ;pirql=6, piv=$16
        LDD     #$0101
        STD     PITR            ;set the periodic timer at 62.5msec
        ANDP    #$FF1F          ;set interrupt priority to 000

***** QSPI Initialization *****

        LDAA    #$08
        STAA    QPDR            ;output pcs0/ss* to 0 when asserted
        LDAA    #$0F
        STAA    QPAR            ;assign QSM port pins to qspi
                                ;module
        LDAA    #$40E
        STAA    QDDR            ;assign all QSM pins as outputs
                                ;except miso
        LDD     #$8004          ;mstr, womq=cpol=cpha=0
        STD     SPCR0           ;16 bits, 2.10 MHz serial baud rate
        LDD     #$0300          ;interrupt generated, no wrap mode
        STD     SPCR2           ;newqp=0, endqp=3, queued for 4
                                ;trans
```

```
*****   Fill QSPI Command.ram to write the config registers of the 14489

        LDAA    #$CE
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no
                                ;delays needed
        STAA    CR1
        STAA    CR2
        LDAA    #$4E
        STAA    CR3             ;cont=0, bitse=1, pcs0=0, no
                                ;delays needed

*****   Fill QSPI Transmit.ram to write the config registers of the 14489

        LDAA    #$3F
        STD     TR0+1           ;store $3F to tran.ram registers
        STD     TR2
        STD     TR3+1

*****   Turn on the QSPI, this will write to the config registers of the
*****   MC14489 drivers

GO      LDD     #$8404
        STAA    SPCR1           ;turn on spi
SPIWT   LDAA    #$80            ;after sending data we wait until the
        ANDA    #$80            ;spif bit is set, before we can send
                                ;more
```

```
        CMPA    #$80            ;check for spi done
        BNE     SPIWT

*****   Fill QSPI Command.ram to write the display registers of the 14489

        LDAA    #$CE
        STAA    CR0             ;cont=1, bitse=1, pcs0=0, no
                                ;delays needed
        STAA    CR1
        LDAA    #$4E            ;cont=0, bitse=1, pcs0=0, no
                                ;delays needed
        STAA    CR2
        STAA    CR4
        LDAA    #$8E            ;cont=1, bitse=0, pcs0=0, no
                                ;delays needed
        STAA    CR3

*****   Fill QSPI Transmit.ram for display registers of the 14489
*****   The beginning LED values will be $00, all of the LEDs will be off

        LDD     #$8000
STD     TR0                     ;TR0 = $8000
STAA    TR3+1                   ;TR1 = $0080
LDD     #$0080                  ;TR2 = $0000
STD     TR1                     ;TR3 = $XX80
        CLRD                    ;TR4 = $0000
        STD     TR2
```

```
        STD     TR4
        LDD     #$0400          ;display registers need 5
                                ;transmissions
        STD     SPCR2           ;newqp=0, endqp=4

*****   ADC Initialization  *****

        LDD     #$0000
        STD     ADCMCR          ;turn on ADC
        LDD     #$0003
        STD     ADCTL0          ;8-bit, set sample period

*****   Initialize the extension registers for the internal ram in bank F
*****   Set up the extension registers to point to bank F

        LDAB    #$0F            ;load b with $0F
        TBEK                    ;transfer Bacc to Ek
        TBXK                    ;transfer Bacc to Xk
        TBYK                    ;transfer Bacc to Yk
        TBZK                    ;transfer Bacc to Zk
        JMP     $F0000          ;jump to internal ram for speed!

*****   Start of Internal 1K RAM

        ORG     $F0000
RAM     CLR     CNT             ;clear LED update counter
        CLR     PK_125          ;clear 125 peak value
```

```
        CLR     PK_500      ;clear 500 peak value
        CLR     PK_1K       ;clear 1k peak value
        CLR     PK_4K       ;clear 4k peak value
        CLR     PK_10K      ;clear 10k peak value

*       Initialization for DSP

        ORP     #$0010      ;set saturation mode for Macc
        CLRD                ;clear Dacc
        TDMSK               ;no modulo addressing
LP      LDY     #COEFBS     ;4 load y with the coef base addr
        LDX     #YTRMBS     ;4 load x with the yterm base addr
        LDHI                ;8 load h and i multiplier and
                            ;multiplicand
        CLRD                ;2 clear Dacc
        STD     ADCTL1      ;6 single 4 conversion, single
                            ;channel AD0

*       Divide input x(n) by 2, no overflow problem

T_OUT   LDAA    LJSRR0      ;6 load Aacc with left jus signed
                            ;ADC value
T_IN    ASRA                ;2 divide by 2
        STAA    AD          ;6 store divide by 2 adc value away

*       Check if LEDs need updating
```

```
        ADDA    #1          ;2 add 1 to Aacc
        STAA    CNT         ;6 store new count
        BNE     F125        ;6,2 check to see if its time to
                            ;update
                            ;the LEDs, time = 256 * 668 cycles
                            ;668 cycles = 40.08usec
                            ;so LED update time is 10.26msec
        LDD     #8404       ;6 load up Dacc
        STD     SPCR1       ;6 turn on QSPI, send LED data out

***** Start of the 125 Hz routine

F125    CLRM                ;2 clear Macc
        LDE     AD          ;6 load Eacc with AD

*       Digital processing algorithm

        TED                 ;2 transfer Eacc to Dacc
        SUBD    XN2_125     ;6 Dacc =  x(n) - x(n-2)
        STD     X_2_125     ;6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_125     ;6 load Dacc with x(n-1)
        STED    XN1_125     ;8 store x(n) to x(n-1) and
                            ;store x(n-1) to x(n-2)
        MAC     2,2         ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2         ;12 beta*(yn2)+Macc=Macc
        MAC     2,2
        TMER                ;6 transfer Macc to Eacc, round for
```

```
        TMER                    ;6 transfer Macc to Eacc, round for
                                ;converg
        ASLE                    ;2 multiply Eacc by 2

*       Get LED encode value from look-up table

        TED                     ;2 transfer Eacc to Dacc
        STAA    LD125+3
        NOP                     ;2 no operation, due to CPU
                                ;pipeline
        NOP                     ;2 no operation, due to CPU
                                ;pipeline
LD125   LDAA    LED_TBL         ;6 load Aacc with the encoded LED
                                ;value from scaled peak LED table

*       Update peak value if needed

        CMPA    PK_125          ;6 compare value to previous peak
                                ;value
        BLS     DN125           ;6,2 branch if not more than peak
                                ;value
        STAA    PK_125          ;6 store new peak value
        STAA    TR4+1           ;6 store new value to 125 qspi
                                ;tran.ram

*       Update y(n-1) and y(n-2)
```

```
DN125   LDD     YN1_125     ;6 load Dacc with y(n-1)
        STED    YN1_125

*****  Start of the 500 Hz DSP routine

F500    CLRM                ;2 clear Macc
        LDE     AD      ;6 load Eacc with AD

*       Digital processing algorithm

        TED                 ;2 transfer Eacc to Dacc
        SUBD    XN2_500     ;6 Dacc = x(n) - x(n-2)
        STD     X_2_500     ;6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_500     ;6 load Dacc with x(n-1)
        STED    XN1_500
        MAC     2,2         ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2         ;12 beta*(yn2)+Macc=Macc
        MAC     2,2
        TMET                ;2 transfer Macc to Eacc, truncate
        ASLE                ;2 multiply Eacc by 2

*       Get LED encode value from look-up table

        TED                 ;2 transfer Eacc to Dacc
        STAA    LD500+3
        NOP                 ;2 no operation, due to CPU
                            ;pipeline
```

```
                NOP                             ;2 no operation, due to CPU
                                                ;pipeline
LD500           LDAA    LED_TBL                 ;6 load Aacc with the encoded LED
                                                ;value from scaled peak LED table

*       Update peak value if needed

                CMPA    PK_500                  ;6 compare value to previous peak
                                                ;value
                BLS     DN500                   ;6,2 branch if not more than peak
                                                ;value
                STAA    PK_500                  ;6 store new peak value
                STAA    TR4                     ;6 store new value to 500 qspi
                                                ;tran.ram

*       Update y(n-1) and y(n-2)

DN500           LDD     YN1_500                 ;6 load Dacc with y(n-1)
                STED    YN1_500

*****   Start of the 1k Hz routine

F1K             CLRM                            ;2 clear Macc
                LDE     AD              ;6 load Eacc with AD

*       Digital processing algorithm
```

```
                TED                     ;2 transfer Eacc to Dacc
                SUBD    XN2_1K          ;6 Dacc = x(n) - x(n-2)
                STD     X_2_1K          ;6 store Dacc to [x(n) - x(n-2)] addr
                LDD     XN1_1K          ;6 load Dacc with x(n-1)
                STED    XN1_1K          ;8 store x(n) to x(n-1) and
                                        ;store x(n-1) to x(n-2)
                MAC     2,2             ;12 gamma*(yn1)+Macc=Macc
                MAC     2,2             ;12 beta*(yn2)+Macc=Macc
                MAC     2,2
                TMET                    ;2 transfer Macc to Eacc, truncate
                ASLE                    ;2 multiply Eacc by 2

*       Get LED encode value from look-up table

                TED                     ;2 transfer Eacc to Dacc
                STAA    LD1K+3          ;
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
LD1K            LDAA    LED_TBL         ;6 load Aacc with the encoded LED
                                        ;value from scaled peak LED table

*       Update peak value if needed

                CMPA    PK_1K           ;6 compare value to previous peak
                                        ;value
```

```
        BLS     DN1K        ;6,2 branch if not more than peak
                            ;value
        STAA    PK_1K       ;6 store new peak value
        STAA    TR2+1       ;6 store new value to 1k qspi
                            ;tran.ram

*       Update y(n-1) and y(n-2)

DN1K    LDD     YN1_1K      ;6 load Dacc with y(n-1)
        STED    YN1_1K

***** Start of the 4k Hz routine

F4K     CLRM                ;2 clear Macc
        LDE     AD          ;6 load Eacc with AD

*       Digital processing algorithm

        TED                 ;2 transfer Eacc to Dacc
        SUBD    XN2_4K      ;6 Dacc = x(n) - x(n-2)
        STD     X_2_4K      ;6 store Dacc to [x(n) - x(n-2)] addr
        LDD     XN1_4K      ;6 load Dacc with x(n-1)
        STED    XN1_4K

        MAC     2,2         ;12 gamma*(yn1)+Macc=Macc
        MAC     2,2         ;12 beta*(yn2)+Macc=Macc
        MAC     2,2
```

```
                TMET                    ;2 transfer Macc to Eacc, truncate
                ASLE                    ;2 multiply Eacc by 2

*       Get LED encode value from look-up table

                TED                     ;2 transfer Eacc to Dacc
                STAA    LD4K+3
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
LD4K            LDAA    LED_TBL         ;6 load Aacc with the encoded LED
                                        ;value from scaled peak LED table

*       Update peak value if needed

                CMPA    PK_4K           ;6 compare value to previous peak
                                        ;value
                BLS     DN4K            ;6,2 branch if not more than peak
                                        ;value
                STAA    PK_4K           ;6 store new peak value
                STAA    TR2             ;6 store new value to 4k qspi
                                        ;tran.ram

*       Update y(n-1) and y(n-2)

DN4K            LDD     YN1_4K          ;6 load Dacc with y(n-1)
```

```
                STED        YN1_4K          ;8 store Eacc to y(n-1), Dacc to
                                            ;y(n-2)

***** Start of the 10k Hz routine

F10K            CLRM                        ;2 clear Macc
                LDE         AD              ;6 load Eacc with AD

*   Digital processing algorithm

                TED                         ;2 transfer Eacc to Dacc
                SUBD        XN2_10K         ;6 Dacc =  x(n) - x(n-2)
                STD         X_2_10K         ;6 store Dacc to [x(n) - x(n-2)] addr
                LDD         XN1_10K         ;6 load Dacc with x(n-1)
                STED        XN1_10K         ;8 store x(n) to x(n-1) and store
                                            ;x(n-1) to x(n-2)
                MAC         2,2             ;12 gamma*(yn1)+Macc=Macc
                MAC         2,2             ;12 beta*(yn2)+Macc=Macc
                MAC         2,2
                TMET                        ;2 transfer Macc to Eacc, truncate
                ASLE                        ;2 multiply Eacc by 2

*   Get LED encode value from look-up table

                TED                         ;2 transfer Eacc to Dacc
                STAA        LD10K+3
```

```
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
                NOP                     ;2 no operation, due to CPU
                                        ;pipeline
LD10K           LDAA    LED_TBL         ;6 load Aacc with the encoded LED
                                        ;value from scaled peak LED table

*       Update peak value

                CMPA    PK_10K          ;6 compare value to previous peak
                                        ;value
                BLS     DN10K           ;6,2 branch if not more than peak
                                        ;value
                STAA    PK_10K          ;6 store new peak value
                STAA    TR1             ;6 store new value to 10k qspi
                                        ;tran.ram

*       Update y(n-1) and y(n-2)

DN10K           LDD     YN1_10K         ;6 load Dacc with y(n-1)
                STED    YN1_10K         ;8 store Eacc to y(n-1), Dacc to
                                        ;y(n-2)
                NOP

END             JMP     LP              ;6 jump back to start another
                                        ;conversion
```

```
***** Exceptions/Interrupts *****
***** This interrupt is used to decrement each LED bar value
***** representing the peak value of each filter band

INT_RT  PSHM    D,CCR       ;stack Dacc and CCR on stack
CK125   LDAA    PK_125      ;load Aacc with 125 peak value
        BEQ     CK500       ;equal to 0?, then CK500
        ANDP    #$FEFF      ;clear C bit
        RORA                ;rotate right once, decrease peak
                            ;value
        STAA    TR4+1       ;store Aacc to 125 Hz qspi tran.ram
        STAA    PK_125      ;store Aacc to 125 Hz peak value
CK500   LDAA    PK_500      ;load Aacc with 500 peak value
        BEQ     CK1K        ;equal to 0?, then CK1K
        ANDP    #$FEFF      ;clear C bit
        RORA                ;rotate right once, decrease peak
                            ;value
        STAA    TR4         ;store Aacc to 500 Hz qspi tran.ram
        STAA    PK_500      ;store Aacc to 500 Hz peak value
CK1K    LDAA    PK_1K       ;load Aacc with 1k peak value
        BEQ     CK4K        ;equal to 0?, then CK4K
        ANDP    FEFF        ;clear C bit
        RORA                ;rotate right once, decrease peak
                            ;value
        STAA    TR2+1       ;store Aacc to 1k Hz qspi tran.ram
        STAA    PK_1K       ;store Aacc to 1k Hz peak value
CK4K    LDAA    PK_4K       ;load Aacc with 4k peak value
```

```
        BEQ     CK10K       ;equal to 0?, then CK10K
        ANDP    #$FEFF      ;clear C bit
        RORA                ;rotate right once, decrease peak
                            ;value
        STAA    TR2         ;store Aacc to 4k Hz qspi tran.ram
        STAA    PK_4K       ;store Aacc to 4k Hz peak value
CK10K   LDAA    PK10K       ;load Aacc with 10k peak value
        BEQ     UPDATE      ;equal to 0?, then UPDATE
        ANDP    #$FEFF      ;clear C bit
        RORA                ;rotate right once, decrease peak
                            ;value
        STAA    TR1         ;store Aacc to 10k Hz qspi tran.ram
        STAA    PK_10K      ;store Aacc to 10k Hz peak value
UPDATE  LDD     #$8404      ;load up Dacc
        STD     SPCR1       ;turn on QSPI, send LED data out
DONE    PULM    D,CCR       ;pull Dacc and CCR from stack
        RTI                 ;return from interrupt

***** Location of start of level 6 interrupt, has to be in bank 0

        ORG     $A000
JMPINT  JMP     INT_RT
```

# BIBLIOGRAPHY

1. Peatman, John B. Design with Microcontrollers. New York: McGraw-Hill, 1988

2. "Technical Summary - 16-Bit Modular Microcontroller". Phoenix, Arizona: Motorola Inc., 1992.

3. Brophy, J. J. Basic Electronics for Scientists. New York: McGraw-Hill, 1966.

4. Lancaster, D. Active Filter Cookbook. Indianapolis: Howard W. Sams & Co., Inc., 1975.

5. Oppenheim, A. V., and Schafer, R.W. Digital Signal Processing. New Jersey: Prentice-Hall, 1975.

6. Rabiner, L. R., and Gold, B. Theory and Application of Digital Signal Processing. New Jersey: Prentice-Hall, 1975.

7. Strawn, J., et al. Digital Audio Signal Processing - An Anthology. New York: William Kaufman, 1985.