Simple Experiments Involving External Control of Algorithm Parameters for an EET Undergraduate DSP Course

Anthony J. A. Oxtoby, Gerard N. Foster
Purdue University, West Lafayette/Kokomo

Abstract

This paper presents an overview of the structure and content of an EET undergraduate course in DSP in which the implementation of application algorithms on a 16 bit fixed point processor is emphasized. Laboratory activities in the course are enhanced by the use of some simple circuitry that interfaces 8 linear slide potentiometers to the data memory bus of the processor thus providing the user with some external control over selected parameters in a given DSP algorithm. A description of the circuitry is given along with examples of experiments featuring the control of delay time and attenuation in audio effects algorithms, the offset, amplitude, frequency and phase in a wavetable based function generator and finally the dB gain of a FIR digital filters.

I. Introduction

The role of DSP in the EET baccalaureate degree at Purdue University is as a capstone course to the sequence of required courses covering analog electronics, digital electronics and introductory microprocessors. The primary aim of the DSP course is to develop the student's understanding of the fundamental concepts, language and some applications of discrete time signal processing whilst also providing the experience of programming a representative programmable DSP processor. It also serves to bring together disparate topics covered in the classes listed above. The main segments of the course are:

- The architecture, instruction set, and software and hardware tools associated with the ADSP-2101 16 bit fixed point programmable digital signal processor.
- Binary arithmetic and fixed point number formats.
- Linear and modulo indirect addressing, circular buffers and the operation of delay lines, wave-tables and look-up tables.
- Serial and memory mapped I/O interfacing.
- Properties of and operations on discrete time signals.
- Introduction to discrete-time systems, linear convolution, z transform and difference equations.
- The Fourier transform, DFT and FFT and an introduction to spectrum estimation.
- FIR and IIR filters – design and implementation.

The course is essentially divided in to two parts with the first half focusing on the processor and the second covering more of the theoretical underpinning leading to the implementation of some common DSP applications such as filtering and the DFT/FFT. Emphasis is placed on the use of the processor hardware and development tools and on the technology of DSP. Thus the course shows the use of common methods of analysis and design used in DSP applications and follows through with their algorithmic implementation. Strong laboratory support is used to reinforce and clarify concepts covered in the classroom through the real time implementation of applications coupled with the use of simulation and emulation tools to perform design, prediction of performance and de-bugging.

II. DSP Laboratory Development Software and Hardware

Most experiments are designed around the Analog Devices ADSP2101 16 bit fixed point microcomputer which has 2k x 24 bit program memory (PM), 1k x 16 bit data memory (DM), both on chip, and operates with a 60ns instruction cycle. Each laboratory station has an ADSP2101 based EZ-LAB[1] hardware platform together with an EZ-ICE in circuit emulator on which all real time code execution and interfacing lab activities are implemented. The EZ-LAB consists of a microcomputer together with an 8 page 64k x 8 bit external boot EPROM containing demonstration programs. A single channel of analog I/O interfacing is implemented through one of the processor's high-speed synchronous serial ports via a CODEC designed for voice band signals and operated at a sampling rate of 8kHz. In addition a 4 channel double buffered 8 bit DAC is mapped onto the processor's data memory data bus. Connection of additional application specific hardware can be made through the processor's two serial ports or mapped onto either memory space via a 60-way expansion connector. The ADSP2101 based hardware was chosen since it is a freestanding unit allowing easy access for adjustment and de-bugging of the hardware and also easy interfacing of external breadboard circuitry. In addition, the assembly language instruction set is relatively high level, and many instructions can be almost intuitively understood. This has proved to be a singular advantage in this introductory course.

The processor specific software development tools consist of an ANSI C compiler, assembler, linker, builder, PROM splitter, simulator and PC-to-emulator communication software. All are installed on a PC linked serially to the EZ-ICE. Executable code can be debugged and run in non-real time on the ADSP2101 simulator which has features for accessing external data files to simulate I/O data transfer and displaying memory contents and I/O data graphically. Alternatively code can be downloaded to the EZ-ICE, which is plugged into the EZ-LAB, and then run in real time. Code can also be examined in the emulator and debugged using single step, multiple step and breakpoint features.

Besides the processor specific software tools, the Windows based software package PCDSP[2] is used to implement more general DSP operations including digital filter design and analysis, convolution, correlation, DFT and FFT spectral estimation and generation and manipulation of discrete time signals. EXCEL spreadsheet software is also used, wherever possible, to force students to work with basic equations associated with DSP operations and applications yet use, and appreciate, the power of the spreadsheet to solve these equations, graphically display results

and quickly implement design changes whilst avoiding the tedium of repetitive calculations. For example it is used to implement the recursive equation associated with the digital sine wave generator prior to implementing it on the processor and thus the effect on the waveform of using different coefficients and initial conditions can be investigated. Also various window functions and the coefficients of FIR filters designed using window methods can easily be generated on the spreadsheet. The frequency response of the filter can then be obtained by using the radix-2 FFT built into EXCEL to perform a DFT on the filter's impulse response coefficients.

III. Indirect Linear and Modulo Addressing

The indirect mode of memory addressing has two forms, linear and modulo addressing, defined below. Addresses are generated automatically in dedicated data address generators or DAGs.

Linear form of indirect addressing: $Ix_{[n+1]} = Ix_{[n]} + Mx$

Modulo form of indirect addressing: $Ix_{[n+1]} = (Ix_{[n]} + Mx - B) \text{mod} Lx + B$

Where: Ix = address pointer register, unsigned 14 bit
Lx = buffer length register, unsigned 14 bit
Mx = modifier register, signed 14 bit
B = buffer base address
X = 0,1…..7

The modulo form of addressing involves the incremental movement of any of the 8 address pointers (Ix) around a memory segment such that the segment appears circular in nature. This form is the one most commonly used in DSP algorithms. It is thus crucial that students have a clear understanding of modulo addressing, and that they develop the ability to follow the movement of the pointers during code de-bugging and also when learning the operation of the more complex algorithms covered in the second half of the course. The principal uses of circular buffers and modulo addressing in this course are:

- For sequential fetching of data from a table of constants such as:
  - The sample values from a wave-table in signal generation
  - The sample values from a cosine wave-table to generate the complex exponentials W_N^{kn} for use in DFT algorithms and twiddle factors W_N^{k} for use in FFT algorithms.
  - The window coefficients in a DFT or FFT algorithm.
  - The impulse response coefficients of a FIR filter.
  - The coefficients of recursive equations such as in an IIR filter algorithm.

- For sequential loading of blocks of sampled signal data for subsequent DFT/FFT processing
- For sequential loading of sampled data into a delay line and extracting past and present samples as required in the implementation of convolution, FIR filters, audio effects etc.
- For loading of input and output data into delay lines and fetching past and present samples as used in the implementation of IIR filters, audio effects, recursive signal generation etc.

The list above essentially consists of two categories viz.:

- Sequential fetching of constant coefficients from a table of values
- Loading/updating and fetching signal data in and out of a circularly accessed segment of memory

Laboratory exercises have been devised in order to demonstrate each of the two categories separately. The introductory exercises on wave-table based generation of periodic signals addresses the first category, whilst the exercise on the generation of echo and reverberation provides experience with manipulating data into and out of a delay line. This then sets the scene for the simultaneous implementation of these two uses of circular buffers first encountered later in the course in the exercises on real time convolution and FIR filter realization and then again for IIR and DFT implementation.

IV. Introductory Laboratory Exercises Associated With Circular Buffers

1. Wave-table Based Signal Generation

This 3 hour set of laboratory exercises is the first to involve circular buffers, modulo addressing and the use of the on board 24 bit timer and occurs in the 4th week of the course following exercises using the ALU, MAC and barrel shifter. It is also the first opportunity for students to implement code on the EZLAB/EZICE hardware, since previous exercises were all performed on the simulator.

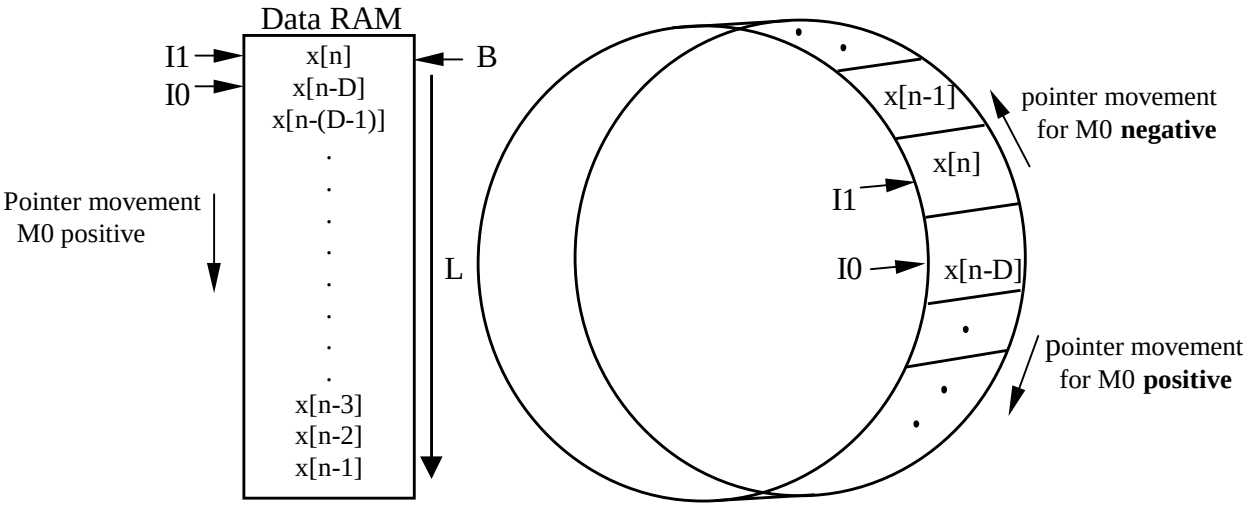
All exercises are based around the retrieval of samples from a wave-table consisting of 256 samples of a single cycle of a sinewave[3]. The aim is to familiarize the student with the initialization of pointer registers, the associated length registers, and modifier registers, which determine the size and direction of the address increments. Thus the first exercises center around the generation of a sine wave of the form $x[n] = A\sin(\omega_d n + \phi)$. Students investigate the relationship between the factors that control the frequency $f_{sig} = \frac{M \cdot f_{clk}}{N \cdot D}$ of the output sine wave.

Where: f_{clk} is the processor clock frequency,
M is the modifier register value ($0 \le M < D$),
D is the length of the wave-table,
N is the effective 24 bit initial value of the timer registers.

In the subsequent exercises the student is required to use the wave-table as the basis for generating other waveforms of specified shape, frequency and amplitude. For example, complex waveforms are synthesized based upon the generation and summation of harmonics differing in amplitude, frequency and phase but all derived from the basic wave-table. This requires the use of multiple pointers initialized to different positions in the wave-table working with a range of modifier values. It also includes the use of the MAC for amplitude scaling and summation.

2. Simple Audio Effects Generation

These exercises occupy one half of one lab period (1.5 hours) and cover the use of circular buffers for streaming sampled data through a delay line an understanding of which is essential to the operation and implementation of FIR and IIR filters as well as many audio effects. Figure 1 below shows the arrangement of a circular buffer used to generate a time-delayed version of a real input signal. In the figure x[n-D] represents the signal sample with the longest delay (D). Thus to generate a 125mS delay with a sampling rate of 8kHz. D must be 1000 and the buffer must have a minimum length of 1001. In this example the pointer I1 is always associated with the address into which the latest sample, x[n], is stored and pointer I0 is used to point to a particular delayed sample which is to be extracted from the delay line. Upon receiving a new input sample x[n] the previous oldest sample x[n-D] is overwritten with x[n] and I1 incremented ready for the next input sample. Thus when a new sample is stored all previous samples in the delay line effectively "age" by one sample period.



Circular Buffer Movement of Pointers Around the Circular Buffer

Modulo addressing $Ix_{[n+1]} = (Ix + Mx - B) \bmod Lx + B$

Figure 1. A Circular Buffer Delay Line

With this exercise students the operation of the delay line is explained and a program developed to produce an output y[n] which is a delayed version of the input x[n] i.e. generate y[n] = x[n-D].

Based upon this code, students develop programs to generate echo and reverberation, as defined below, and investigate the effects of changing α and D.

Slap echo: $y[n] = x[n] + \alpha x[n-D]$

Simple reverberation: $y[n] = x[n] + \alpha y[n-D]$

V. Fixed Point Binary Arithmetic and Formats

One of the more taxing aspects of the course for students is the implementation of binary arithmetic. The multiply/accumulate operation is central to so many DSP algorithms and the risk of overflow is always present on a fixed-point processor. Time must be taken to determine the likely range of data values generated in a particular algorithm and to select an appropriate format and apply proper scaling to signal data and algorithm coefficients[4].
The most common 16 bit 2's complement fixed point format used is 1.15 fractional with an equivalent decimal range of –1.0 to + 0.9999…… In fact the default mode on this processor is fractional mode in which an automatic left shift by 1 bit is implemented on the 2.30 result produced from two 1.15 multiplicands, to produce a 1.31 result. The 16bit 1.15 result can then be extracted from the MS bits of the 32 bit result and stored or re-used in another operation. No product overflow can occur using fractional binary format except, of course, during a succession of additions or the unlikely case of $(-1.0)^2$. Intermediate overflows can be tolerated in the 2's complement system however, provided the final result is within the range of the 2's complement format i.e. provided the sign extension of the final result is valid.

The 16bit integer signed and unsigned formats are also used, in DAG registers and timers for example, as are other signed formats, albeit less commonly, according to the demands of the algorithm. In addition some form of scaling is usually necessary with ADC and DAC data.

The class is introduced to working with different binary formats after the introduction to the computational units and exercises are executed on the simulator. However, in order to expose the students to simple situations where some scaling of binary data into another format is necessary, the slide potentiometer was introduced to permit the control of parameters associated with the previous exercises involving the manipulation of delay lines and wave-tables. It also provides an example of hardware interfacing and more experience with pointer manipulation. The scaling is based on the application of simple zero and spanning techniques.

VI. Slide Potentiometer Interface Circuit

The simple circuit, shown in figure 2, interfaces the MAX161 8 channel data acquisition system to the ADSP-2101 EZ-LAB and has been designed to provide the user with the means to adjust the most significant 8 bits of 8 x 16 bit data memory locations using 8 slide potentiometers. Eight 0-5v analog voltages are multiplexed to the 8 bit ADC and the resulting binary data is then stored in the on-board 8byte dual port RAM. The MAX161 features tri-state I/O and contention logic. Address lines A0, A1 and A2 are used to select the channel to be read from the RAM which has been fully decoded onto external data memory at 8 consecutive locations from DM 0x0000 through DM 0x0007. The ADC is driven from a 1MHz. clock which sets the sampling rate for each channel to around 1.5kHz. Raw data into the ADSP-2101 has a range 0x00FF to 0xFFFF. Timing requirements for the MAX161 are such that the chip select (CS) signal must remain low for a minimum of 250ns during a memory read. Thus, with a processor clock period of 60ns a total of 4 periods (240ns) are required to reliably read a single 16bit word from the potentiometer board (1 instruction period plus 3 wait states).

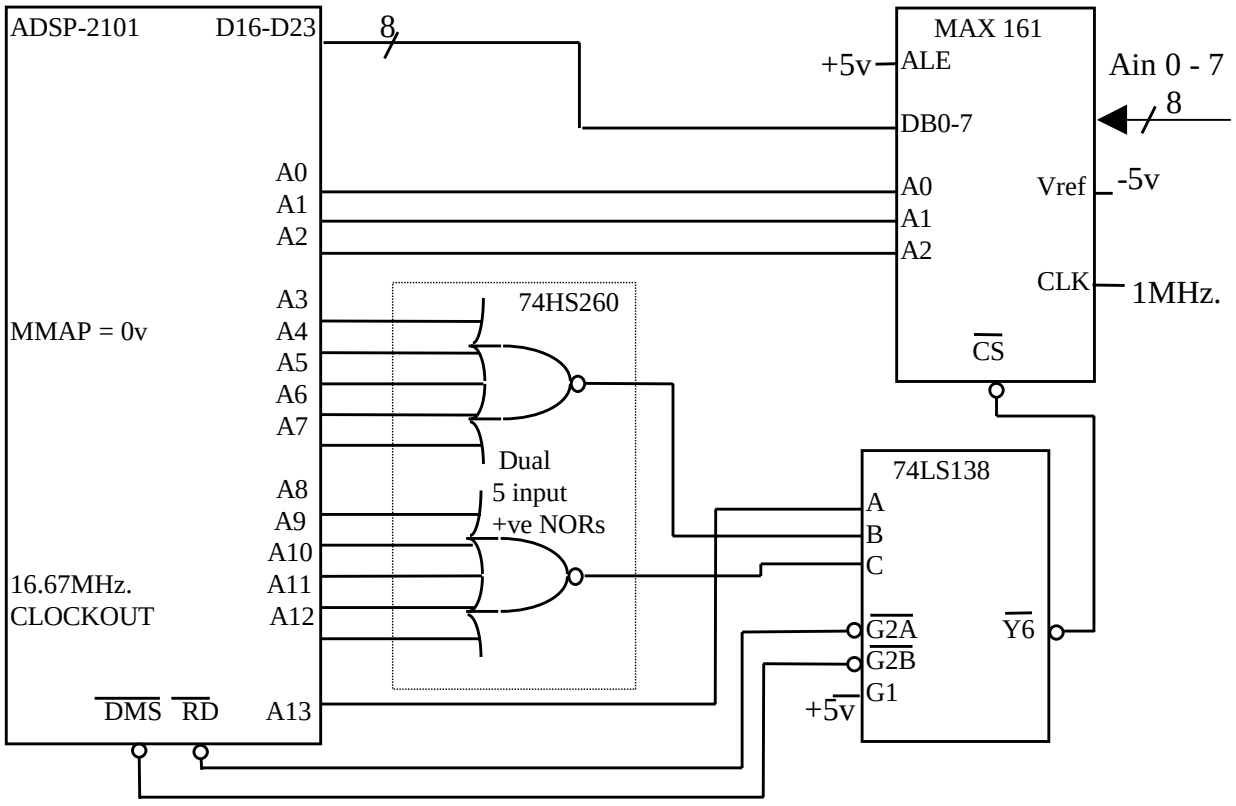


Figure 2. Slide Potentiometer to ADSP-2101 Interfacing Circuit

VII. Introductory Laboratory Exercises With External Control

1. Wave-table Based Signal Generation With External Control

This exercise occupies 1.5 lab periods (4.5 hours) and represents the first encounter the students have with the slide potentiometer board. The basic exercise is to write an algorithm to generate two sine waves that have variable offset, amplitude, phase ($\pm\pi$) and fine and course frequency control each of which is derived from an 8 bit binary number determined by the position of a particular potentiometer. Before tackling the generator problem the students are given a simple program which continuously reads the raw binary data from the 8 potentiometers and then converts them to 2s complement numbers. This software serves double duty as a hardware check and a starting point for later code to be generated by the student.

The function generator problem is set up for the student in that the essential structure of the program is established and the range of control of each parameter defined along with the means by which it is to be controlled in the processor. Thus we have for the parameters:

- Fine frequency controlled by the 8 LS bits of the timer
- Course frequency controlled by a modifier (M) register with $1 \le M \le 33$ for a 256 point wave-table in order to avoid excessive distortion of the output waveform
- Phase controlled by a modifier register with $-128 \le M \le 128$ for a 256point wave-table.

- Amplitude and offset controls are left to the student's discretion.

Thus the student is faced with the need to scale raw 8 bit unipolar binary data with range 0x00xx to 0xFFxx to unsigned and signed integers for the timer and modifier registers respectively and to 1.15 2's complement fractional numbers for the offset and amplitude control. The xx indicates the undefined states of the lower 8 data lines of the 16bit data memory data bus.

The structure of the program is pre-defined such that reading and scaling of the raw binary data from the potentiometer board must be executed as a background task. Generation of each of the two output signal samples including amplitude and offset adjustment are performed as a foreground task i.e. performed as a timer interrupt routine. Two precautions must be observed here. The timer register contents must have a minimum value in order to allow sufficient time between interrupts to execute the interrupt code segment and be able to reliably read at least one potentiometer in the background task otherwise external control is lost. Thus the student must consider factors such as the number of instruction cycles used in the interrupt routine, including wait states which are included to satisfy the 4 channel DAC timing requirements, interrupt latency and total instruction cycles necessary to read at least one potentiometer. In addition care must be taken to avoid corruption of register contents as the processor switches between tasks. The use of a shadow set of data registers in the ADSP-2101 associated with the ALU, MAC and barrel shifter helps the programmer alleviate this problem.

2. Simple Audio Effects Generation With External Control

This exercise is performed in the second half of the laboratory session on audio effects. Referring to figure 1., in order to control the delay D using a slide potentiometer the pointer I0 must first be aligned with pointer I1 and then offset from I1 using a modifier (M1) register the contents of which are derived from the raw potentiometer data. This operation is in addition to the incrementing of pointer I1 around the buffer as each new sample of the input signal is loaded into the buffer. In this way an initial offset, corresponding to the desired delay, can be set between the pointers. Thus if we tie modifier M1 to the scaled data derived from the slide pot we can dynamically control the delay producing offset between I0 and I1. Note that the modifier value is the negative of the desired delay offset D. To produce a 125ms delay at f_s = 8kHz.for example, pointer I0 must be offset from I1 by 1000 locations backwards in the circular buffer. Thus the contents of modifier M1must first be set to -1000 and then used to modify pointer address I0. The example below shows the use of a potentiometer to control the delay in the output signal from a delay line 1001samples long, operating at a sampling frequency of 8kHz. The modifier M1's contents must be controllable over the range 0 to –1000 in order to control the offset between the input data pointer I1 and the output data pointer I0 over the full range of delay. Thus we must scale as:

Raw data		M1 contents
0xFFxx = 1111 1111 xxxx xxxx	to	-1000_{10} = 0xFC18 (16.0 format)
0x80xx = 1000 0000 xxxx xxxx	to	-500_{10} = 0xFE0C
0x00xx = 0000 0000 xxxx xxxx	to	0_{10} = 0x0000

Note that the raw binary data must ultimately be scaled to 14bit 2's complement integer (14.0) format. A simple approach is to use the barrel shifter to logically shift, avoiding sign extension,

the raw data 1 bit to the right and thus convert the raw data to positive only. The lower 7 bits are then masked to zero, using the ALU, and the result scaled to the desired range by multiplication by -0.030742.. , 0xFC11 in 1.15 format, in the MAC. Thus a 16.0 signed integer is multiplied by a 1.15 fraction with the MAC in fractional mode. An automatic arithmetic shift of the 17.15 product occurs producing a 16.16 result in the MAC product register. The 16 MS bits which form a 16.0 result are then transferred as a valid 14 bit result to the modifier register since the range 0 to -1000_{10} is within the 14.0 range. A summary is given below.

0xFFxx	⇒	0x7F80	⇒	0xFC18 (-1000_{10})	⇒	0x3C18
0x80xx	⇒	0x4000	⇒	0xFE08 (-504_{10})	⇒	0x3E08
0x00xx	⇒	0x0000	⇒	0x0000 (0_{10})	⇒	0x0000

When using the slide potentiometer to control the amplitude of the delayed signal such as with or reverb control $y[n] = x[n] + \alpha y[n-D]$, where $0 \le \alpha \le 0.999...$, the scaling again involves logical shift and masking. The resulting data, however, is treated as a 1.15 fraction in a MAC input register, with y[n-D] fed into another MAC input register. The resulting multiplication is performed in fractional mode. Thus the scaled potentiometer data with range 0x0000 to 0x7F80 is now interpreted as a fractional number with range 0 to 0.992675.

VIII. Real-Time Convolution

The real-time direct form of implementation of FIR filters is based on the implementation of the linear convolution of the input data sequence x[n] with the causal impulse response sequence h[n] of length N according to the convolution sum given by:

$$y[n] = \sum_{k=0}^{N-1} x[k]h[n-k]$$

Thus in this exercise, the student is exposed to the simultaneous manipulation of the delay line and a table of filter coefficients. Two pointers are required to sequentially load the MAC with pairs of data to perform a sum-of-products but the pointers must also implement the time reversal and relative shift indicated in the equation above. The pointer movement is illustrated below in figure 3 for the case of N = 256.

The pointer I4 is used to point to the impulse response data h[n] located in PM and starts and finishes at the same point for one complete convolution cycle. At the end of a convolution cycle, the DM pointer I0 will be pointing to x[n-255], the oldest input sample. When a new sample x[n] is acquired from the ADC, the oldest sample is overwritten with x[n] and the pointer I0 incremented. The pointer is now pointing to the new x[n-255] which is the previous x[n-254] but is now one sample older and thus becomes the ‘new oldest input sample’. The pointer I0 is now incremented in the direction shown during the convolution cycle and finishes the cycle pointing to x[n-255] which is the correct memory position for the next sample from the ADC to be used to overwrite the oldest sample. Thus the time reversal operation in the convolution process is effectively implemented by running the pointer I4 in the reverse direction through the impulse response data in PM, whilst the 1place shift operation after each successive convolution

cycle is accomplished in the DM circular buffer by overwriting the oldest data with the newest sample x[n]. Note that the movement of I0 is in the opposite direction to I4.

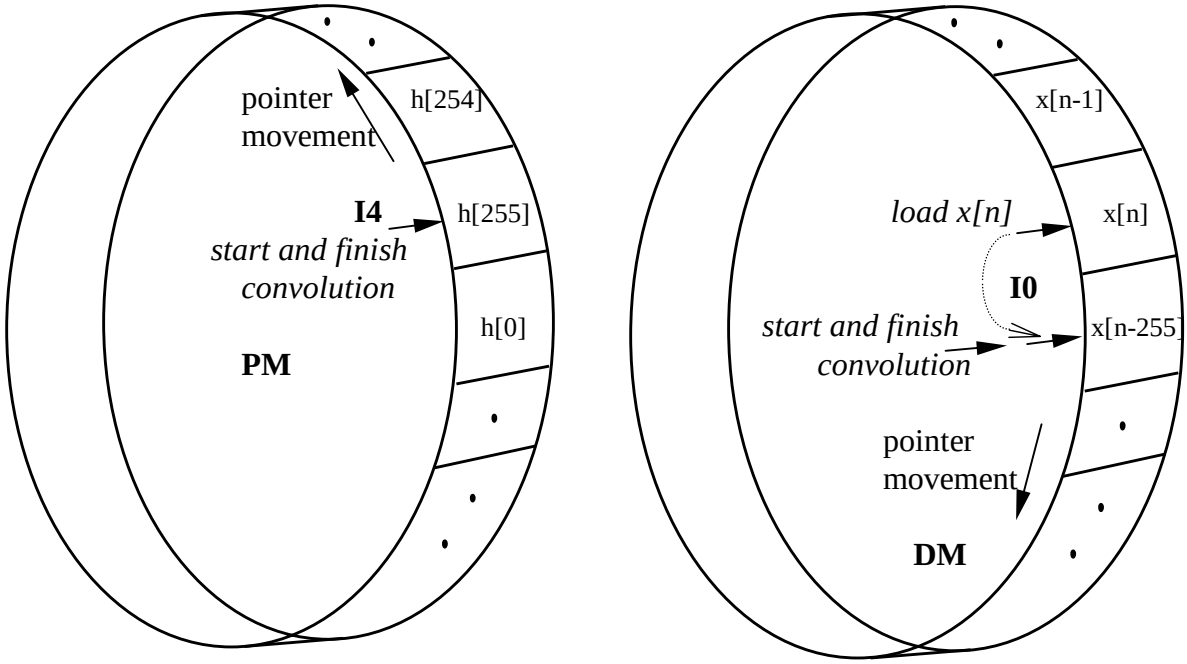


Figure 3. Pointer movement in Real-Time Convolution

IX. Gain Control of a Filter Using a Slide Potentiometer and Look-Up Table

Suppose we have a digital filter and we wish to control the pass band gain of the filter over a specified range of boost and cut using an external potentiometer as shown in figure 4 below. Specifically consider the example shown below which consists of a low pass FIR filter. The relative gain of the filter is shown for the potentiometer positions at maximum (+10dB of boost), minimum (-10 dB of cut) and mid-point (0dB). Gain variation is simply obtained by multiplying the output signal w[n] from the basic FIR filter by a 1.15 number derived from the raw 8 bit data representing the position of the potentiometer. It is thus necessary to scale the raw pot data to a range of 1.15 numbers which will produce the ±10dB of gain variation and, since the potentiometers are linear, to scale so as to establish a logarithmic relationship.
Since the range of positive 1.15 numbers is 0 to 0.99996948…., then the above numbers must re-scaled to maintain the ratios but keep it inside the 1.15 range which effectively means scaling to a gain range of 0 to -20 dB. Three of the 256 values required in the table are given below where address values yy are determined by the base address of the table.

Raw data	⇒	Table address	⇒	dB Gain	⇒	Data table content (1.15)
0xFFxx	⇒	0xyyFF	⇒	0 dB	⇒	0.99996948.. = 0x7FFF
0x80xx	⇒	0xyy80	⇒	-10 dB	⇒	0.316218.. = 0x287A
0x00xx	⇒	0xyy00	⇒	-20 dB	⇒	0.0999969.. = 0x0CCD

In order to achieve the logarithmic variation a 256point look-up table is used, with the potentiometer position being used to determine the point in the table from which the scaled data is fetched. Movement through the look-up table is simply implemented by initializing a pointer to the base address of the table prior to looking up each new value, and then offsetting the pointer address with an increment value (0 - 255) stored in a modifier register and derived from the potentiometer raw data. The increment value is trivially obtained by using the barrel shifter to perform a logical shift by 8 bits to the right on the raw potentiometer data. Generation of the look-up table data is performed using EXCEL. The process simply involves dividing the range 0 to -20dB into 256 increments, converting from dB to numerical gain and then converting these fractional gain values into 1.15 hexadecimal values ready for storage in the look-up table buffer during the linking process.

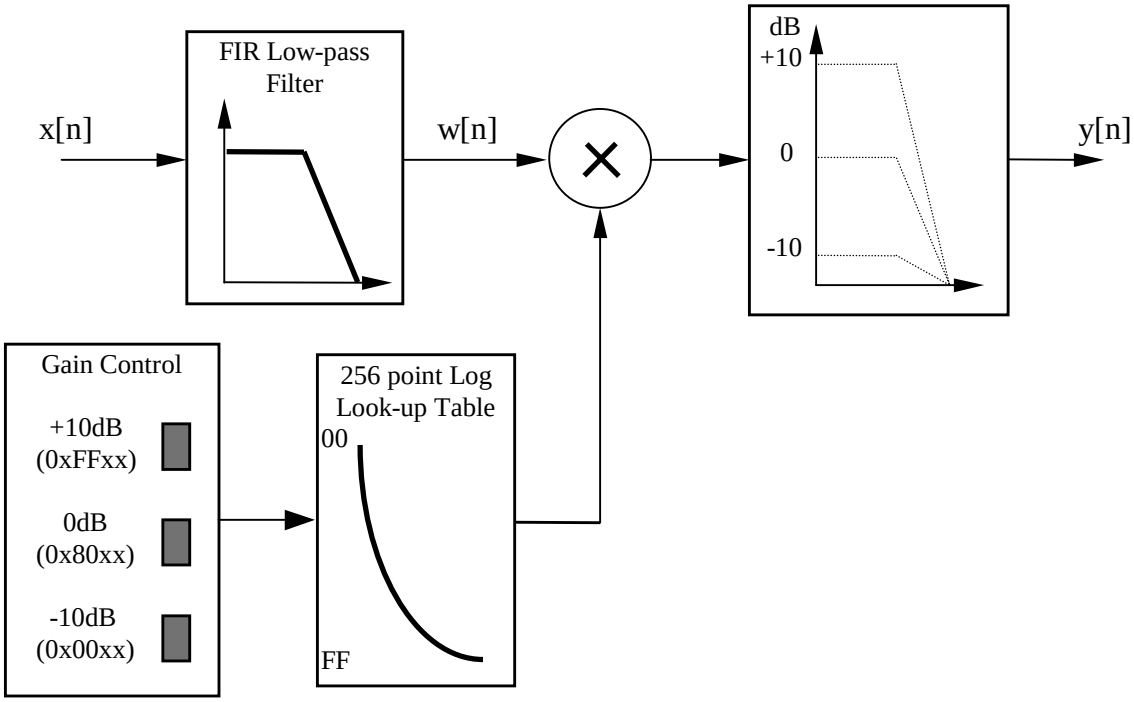


Figure 4. A Single Channel of FIR Filter Gain Control

X. Conclusion

The exercises involving waveform generation and audio effects are implemented at a point in the course when circular buffers and modulo addressing are first introduced and just after some simple programs using the ALU, MAC and barrel shifter instructions have been developed. They are included in order to spur motivation and to provide an opportunity for students to consolidate their understanding of the instruction set and gain confidence in their code writing abilities. In addition students are presented with realistic situations in which they must write routines to apply the techniques they have learned for scaling raw binary data into different formats and ranges necessary to satisfy the roll the scaled data plays in controlling a particular parameter. All of the exercises involve manipulation of pointers used in addressing circular buffers and as such provide experience in pointer initialization, manipulation and especially in

debugging pointer movement. This experience is invaluable in preparing students for the use of pointers in more advanced DSP algorithms such as convolution, correlation, digital filters, and DFT and FFT implementation which appear in the second half of the course. One other benefit is the experience the students gain in working with circuitry, timing and programming issues associated with interfacing external I/O devices to a DSP processor. This experience has been put to use in several senior design projects undertaken after completing this course.

Bibliography

1. Analog Devices, ADSP-2100 Family EZ Tools Manual. 1994.
2. Alkin, O., Digital Signal Processing – A Laboratory Approach Using PCDSP. Prentice-Hall, 1994.
3. Motorola Inc., Digital Waveform Synthesis Using the DSP56001. APR1/D Rev1, 1988.
4. Baudendistel, K., An Improved Method of Scaling for Real-Time Signal Processing Applications, IEEE Trans. Educ., vol37, pp281-88, Aug. 1994.

ANTHONY J. A. OXTOBY

Tony Oxtoby is an Associate Professor of Electrical Engineering Technology at Purdue University in West Lafayette, Indiana. He developed the course in Digital Signal Processing now required at three Purdue campuses. In addition he has presented DSP workshops for industry and academics.

GERARD N. FOSTER

Jerry Foster is an Associate Professor of Electrical Engineering Technology at Purdue University in Kokomo, Indiana. He coordinates and teaches the digital and microcomputer sequence of courses. His areas of interest include microcomputers and DSPs, design projects, and programming in C, C++ and Java. Professor Foster is an active member in ISD (Information Systems Division) of ASEE.