AC 2011-549: TEACHING DIGITAL FILTER IMPLEMENTATIONS USING THE 68HC12 MICROCONTROLLER

Li Tan, Purdue University North Central

DR. LI TAN is currently with the College of Engineering and Technology at Purdue University North Central, Westville, Indiana. He received his Ph.D. degree in Electrical Engineering from the University of New Mexico in1992. Dr. Tan is a senior member IEEE. His principal technical areas include digital signal processing, adaptive signal processing, and digital communications. He has published a number of papers in these areas. He has authored and co-authored three textbooks: Digital Signal Processing: Fundamentals and Applications, Elsevier/Academic Press, 2007; Fundamentals of Analog and Digital Signal Processing, Second Edition, AuthorHouse, 2008, and Analog Signal Processing and Filter Design, Linus Publications, 2009.

Jean Jiang, Purdue University North Central

DR. JEAN JIANG is currently with the College of Engineering and Technology at Purdue University North Central, Westville, Indiana. She received her Ph.D. degree in Electrical Engineering from the University of New Mexico in 1992. Her principal technical areas are in digital signal processing, adaptive signal processing, and control systems. She has published a number of papers in these areas. She has co-authored two textbooks: Fundamentals of Analog and Digital Signal Processing, Second Edition, AuthorHouse, 2008, and Analog Signal Processing and Filter Design, Linus Publications, 2009.

Teaching Digital Filter Implementations Using the 68HC12 Microcontroller

Abstract

We present our pedagogy for teaching digital filter implementations using the 68HC12 microcontroller. In the Electrical and Computer Engineering Technology (ECET) curriculum, a microcontroller has been used as a popular platform for teaching an embedded system course in the sophomore year. After completing the course, students become familiar with the microcontroller software development tools. They are able to program using assembly and C languages, and apply necessary software and hardware interface for hands-on applications. In fact, most microcontrollers are capable of performing basic digital signal processing (DSP) tasks such as digital filtering. Therefore, in this paper, we first proposed a simple DSP platform, which consists of the low-cost 68HC12 microcontroller, a signal condition circuit, and a digital-to-analog converter device for teaching the DSP course. Using the proposed DSP platform, we present our instructional techniques for digital filter implementations and related applications. Our student survey results regarding an adoption of the 68HC12 microcontroller for teaching DSP show that the microcontroller is a cost and learning effective tool and can be used as an alternative when the DSP dedicated hardware is not available.

I. Introduction

In the Electrical and Computer Engineering Technology (ECET) curriculum, a microcontroller has been used as a popular platform for teaching an embedded system course in the sophomore year. After completing the course, students become familiar with the microcontroller software development tools. They are able to program using assembly and C languages and apply necessary software and hardware interface for hands-on applications. In addition, most microcontrollers are capable of performing basic digital signal processing (DSP) tasks such as digital filtering. On the other hand, a low-cost DSP solution is preferred in many DSP applications. For example, a low sampling rate (100 Hz) is fast enough to process temperature signal, light intensity, air pressure, mechanical strain, or seismic signal. Meanwhile, a low analog-to-digital (ADC) resolution (8-bit data) in these applications may be sufficient. Hence, an adoption of a low-cost microcontroller instead of a digital signal processor with full capability is a cost effective choice. Considering these facts, using a microcontroller for a DSP course in the junior year could offer the following benefits to ECET students: (1) a microcontroller can be an alternative and cost effective solution when a DSP processor such as TMS320C67xx is not available; (2) students can save a significant amount of time for learning and familiarizing with the new system architecture, its corresponding development tools and assembly instructions. Instead, they can focus on learning the implementation of digital filters and DSP applications; (3) the microcontroller is flexible to use for various applications including signal processing.

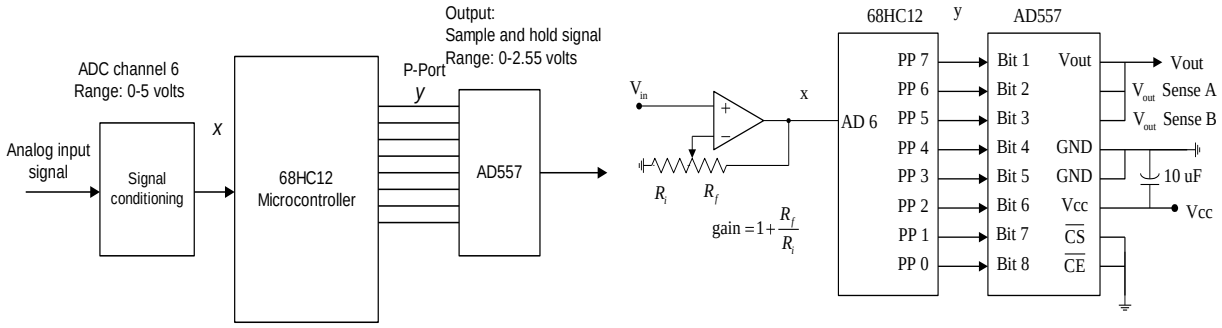
In this paper, we present our pedagogy for teaching digital filter implementations using the 68HC12 microcontroller. The paper is organized as follows: we first describe a simple DSP system that consists of the low-cost 68HC12 microcontroller, a signal condition circuit, and a digital-to-analog (DAC) converter device. A simple program to flexibly set up the sampling rate is then developed and the key assembly instructions for digital filtering are reviewed. Next, we

illustrate the fixed-point data format, finite impulse response (FIR) filter and (infinite impulse response) IIR filter structures with direct forms I and II. After digital filters are designed by using MATLAB, real-time FIR and IIR filter implementations are developed using a linear buffering technique. Finally, we examine student surveys regarding adoption of the 68HC12 microcontroller in their DSP course and discuss the possible improvement based on the survey.

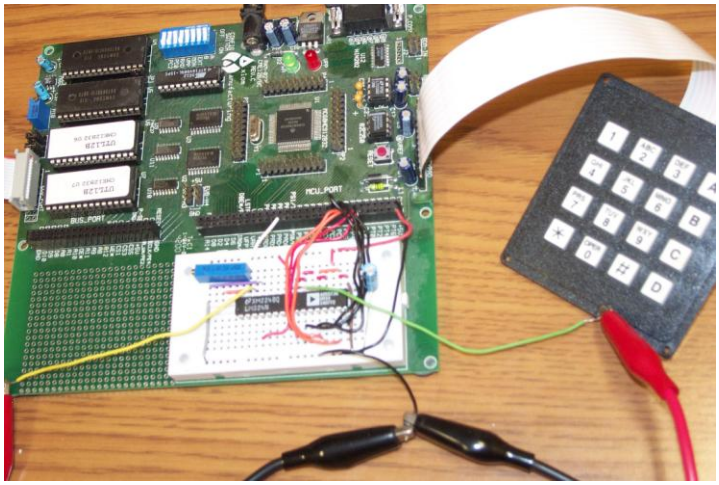
II. DSP Using an HC12 Microcontroller

A. Hardware Setup and Interface

We focus on the development of a simple DSP system by using the 68HC12 microcontroller, which is adopted as a cost and learning effective tool for our DSP course. The detailed information regarding the 68HC12 microcontroller's architecture, interface, and instruction set can be found in the textbook written by D. Pack[1]. Figure 1 depicts our simple DSP system.



(a) Block diagram. (b) Pin assignments.



(c) DSP hardware setup.

Figure 1. A simple DSP system using the 68HC12 microcontroller.

As shown in Figure 1(a), the proposed DSP system consists of the 68HC12 microcontroller, a signal conditioning circuit, and a digital-to-analog (DAC) unit. First, the analog signal from a sensor is conditioned via amplification to fit the voltage range designated between 0 to 5 volts.

The amplified analog signal is then fed to the 68HC12 microcontroller via an ADC channel 6 (any other channel can also be configured) with an 8-bit resolution. The signal conditioning circuit is a standard voltage amplifier, which can be easily designed using an Op-Amp circuit [see Figure 1(b)]. Note that an anti-aliasing low-pass filter is not included, since the platform needs be flexible for various sampling rates set by internal software. The microcontroller processes the converted 8-bit digital value and sends the processed digital output to its 8-bit parallel port. A DAC unit (analog device AD557 circuit) shown in Figure 1(b) applies the 8-bit information and generates a recovered analog signal. Note that the DAC output is essentially a sample and hold waveform which contains image frequencies. These image frequencies can be filtered by a reconstruction low-pass filter (also called the anti-image low-pass filter). Again, a reconstruction low-pass filter is not included due to the flexibility requirement for a sampling rate selection. The DAC output range is between 0 to 2.55 volts. Figure 1(c) displays the DSP hardware setup. For lab experiments, an analog input signal can be produced by a function generator while the analog input and output signals can be examined via an oscilloscope.

B. Software Setup

Considering that students are familiar with the software development tool such as AxIDE (The detailed information can be found in reference[2]), assembly and C languages, and Motorola 68HC12 instruction set, a simple software setup program for real-time signal processing depicted in Figure 2 can first be studied.

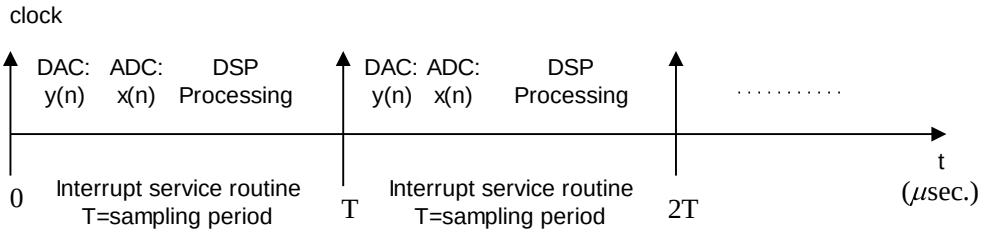


Figure 2. Real-time process using the 68HC12 microcontroller.

As shown in Figure 2, an interrupt driven application is required and an interrupt service routine is executed at the beginning of each sampling period of T seconds. During each sampling period, the microcontroller sends the processed digital output y(n) to the DAC unit, performs ADC conversion to obtain a new digital input data x(n), and processes the digital input x(n) to produce a digital output y(n). The digital output y(n) will be sent out when the next interrupt is activated. Since the microcontroller has a slow clock rate, assembly coding is required in order to maximize the processing capability, that is, maximize the usage of machine execution time. For example, if a sampling rate is set up to 8000 Hz, then the sampling period T equals 125 micro seconds. Again, since the 68HC12 microcontroller has a clock rate of 8 MHz (1 clock tick =1/8MHz=0.125 micro seconds), the maximum number of clock ticks for processing each sample is 1000 (clock cycles). Given the sampling rate, students are required to examine execution time available for each filter implementation to ensure that the total number of clock ticks is within its limit. Figure 3 lists the program segment of sampling, ADC, and DAC conversions. Note that the number of ticks used as 1000 can be changed to set a different sampling rate. Using the machine code information[1], the setup program with a sampling period

of 125 micro-seconds listed in Figure 3 takes 43 clock ticks in total, leaving 957 clock ticks for DSP processing. Of these 43 clock ticks, setting up the interrupt for sampling instant takes 22 clock ticks, sending a digital output requires 6 clock ticks, and performing ADC conversion costs 15 clock ticks. The memory allocation in general is depicted in Figure 4, where data area starts at address 0x2000 while the program area begins at 0x2100. The data area is designated between 0x2000 and 0x2100, in which the input buffer, filter coefficients, and output data buffer reside. Beyond the 0x2100 is the program code area, where the executable codes reside.

Once students verify the sampling program, they can begin to change the sampling rate by themselves. At this stage, using a function generator to produce an input sine wave, they are able to examine the input and output signals on an oscilloscope. In addition, since an anti-aliasing low-pass filter is not included in the DSP system, students are also able to examine the aliasing effects by inputting a sine wave with its frequency value larger than the Nyquist limit (half of the sampling rate).

```
/* data area */
#data    0x2000

/* source code area */
#code    0x2100

filt_oc2()
{
#asm
done     LDAA $008E  ;
       ANDA #$04
       BEQ done
       LDD  $0094
       ADDD #1000   ;  set up  8000 Hz sampling rate: T=1000*125 microseconds
       STD $0094
       LDAA $008E
       ORAA #$04
       STAA $008E
       LDAA $2060     ; load  the processed data
       STAA $0056     ; send the processed data to Port P and DAC device
       LDAA #$26
       STAA $0065   ;  set up the ADC scan mode for AN channel 6
wait     LDAA $0066 ; start ADC, and check the completion flag
       ANDA #$80
       BEQ wait      ; complete ADC conversion
       LDAA $0074  ; obtain the converted from ADC resultant register 2

       ; this area for the DSP algorithm
       STAA $2060  ; store the converted data to the output buffer
#endasm
}

main()
{
  initA2D();
  initOC2();
  while(1) {
    filt_oc2();
  }
```

Figure 3. A sample program segment for sampling rate and ADC setup.

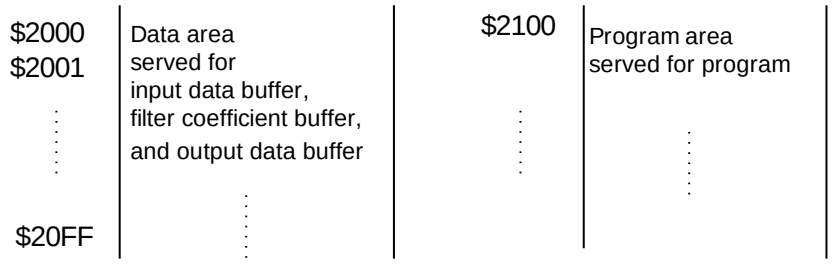


Figure 4. Memory allocation for digital filter implementations.

C. FIR Filter Implementation

Now students can begin to implement a finite impulse response (FIR) digital filter designed using MATLAB according to the given filter specifications. For example, the following designed FIR filter is required to be implemented.

$$y(n) = 0.0060x(n) + 0.0493x(n-1) + 0.1733x(n-2) + 0.25x(n-3)$$
$$+ 0.1733x(n-4) + 0.0493x(n-5) + 0.0060x(n-6)$$

The memory utilization including FIR filter buffers is shown in Figure 5.

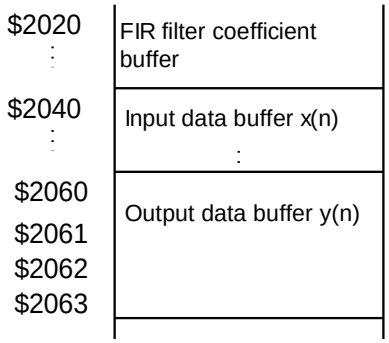


Figure 5. Memory utilization for FIR filter implementation.

Each filter is implemented in a fixed-point format[3-5] in which each data contains 15 bits for magnitude and 1 bit for sign bit (Q-15 format). The 2's complement form is used for any negative number. The designed FIR filter coefficients are quantized into 16 bits as following:

$$b_0 = 0.006 \times 2^{15} \approx 197,$$
$$b_1 = 0.0493 \times 2^{15} \approx 1615,$$
$$b_2 = 0.17331 \times 2^{15} \approx 5679,$$
$$b_3 = 0.25 \times 2^{15} \approx 8192,$$
$$b_4 = 5679,\ b_5 = 1615,\ b_6 = 197$$

In the program (refer to Figure 4), students can enter the quantized filter coefficients in the data area as shown below:

```
-------------------------------------------------------------------------------------------
        #data 0x2020
        int code[7]= {197, 1615, 5679, 8192, 5679, 1615, 197};
-------------------------------------------------------------------------------------------
```

The FIR filtering requires multiplication and accumulation (MAC) operations. Multiplying two 16-bit numbers (Q-15 format) will result in a 31-bit number (Q-30 format). Figure 6 illustrates a fixed-point multiplication using Q-15 formats. After multiplication, the result must be shifted left by 1 bit to form a 32-bit data (Q-31 format).

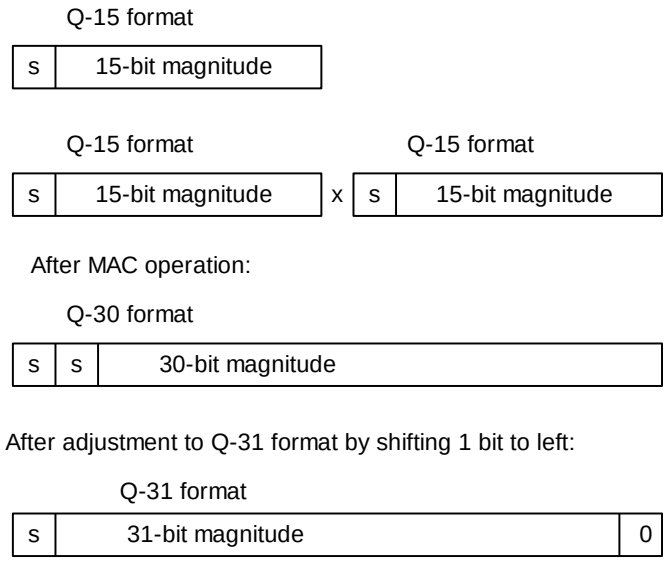


Figure 6. MAC operation in Q-format.

The filtered output data y(n) is stored starting at address 0x2060 with consecutive 4 bytes, whose highest byte will be sent to the parallel port as shown Figure 7. Note that the highest byte is sent out to match the 8-bit ADC resolution.

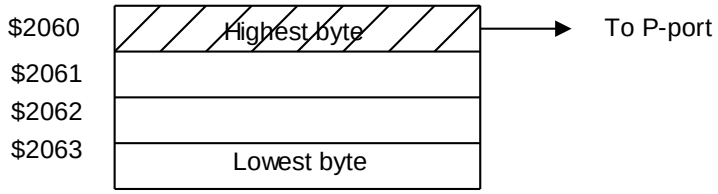


Figure 7. Output data byte for DAC.

Before coding the digital filtering operation, students are instructed to learn how to update input data using an FIFO buffer (first in-first out linear buffering technique) as depicted in Figure 8, where the input buffer ranging from 0x02040 to 0x0204C.

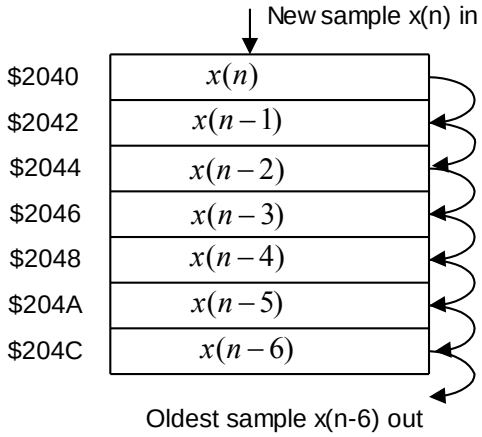


Figure 8. Input buffer using FIFO.

The digital filtering operation is described in Figure 9. As illustrated in the figure, the 68HC12 microcontroller offers a special instruction, called EMACS (multiply and accumulate with the multiplier and multiplicand each having 16 bit signed numbers) operation. The accumulated result is stored as a 32 bit signed number. This instruction requires thirteen (13) clock cycles for execution. It uses index X and index Y registers to hold the addresses of multiplier and multiplicand, respectively. The multiplied result is added to the specified address such as 0x2060 as depicted in Figures 9 and 10. After each "EMACS" operation, the index registers must be increased four (4) times to fetch the next multiplier and multiplicand for the next "EMACS" operation until the filtering processing is completed.

EMACS syntax: [M(x):M(x+1)] x [M(y):M(y+1)] + M~M+3

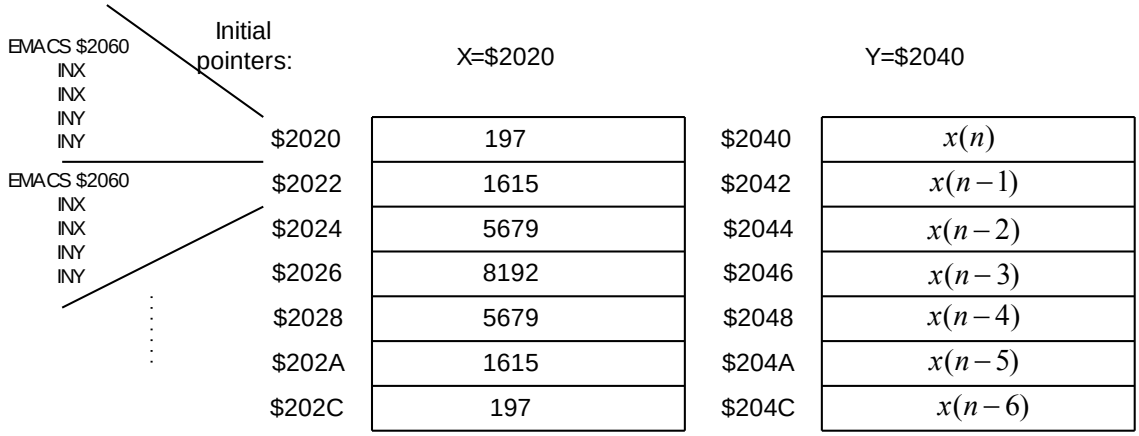


Figure 9. Digital filtering using the EMACS instruction.

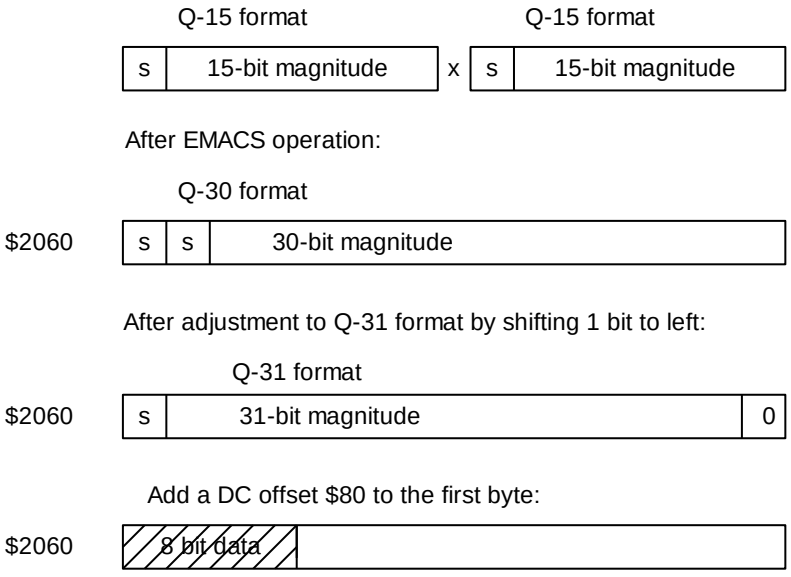


Figure 10. Output data format.

Note that a DC offset value of $80 is added to the first byte (highest byte), since an ADC557 chip only converts a positive number. The final 8-bit data at location 0x2060 is now ready to be sent out to the parallel port. Figure 11 lists a program segment.

```
 ...        ;sampling rate
LDAA $2060  ;output filtered data
STAA $0056
LDD #$00    ; clear the result
STD $2060
STD $2062
…           ; done with ADC
LDD $204A   ;linear buffering
STD $204C
LDD $2048
STD $204A
LDD $2046
STD $2048
LDD $2044
STD $2046
LDD $2042
STD $2044
LDD $2040
STD $2042
LDAA $0074  ;get new input sample
LDAB #$00
ADDA #$80  ;subtract DC offset
STD $2040  ;update new sample
LDX #$2020  ; the first coefficient
LDY #$2040  ; the first data
EMACS $2060 ;
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060 ; done with filtering
LDD $2060
LSLD  ;adjust it to Q-15 format
ADDA #$80  ; add DC offset
STD $2060 ; store to output buffer
```

Figure 11. A sample program segment for FIR filter implementation.

After studying the sample program shown in Figure 11, students are required to examine their own codes to see if the execution time is within its limit set by the sampling rate. Then they program the "EMACS" operation mechanically. Once the program is successfully implemented, students are required to replace the repetition part of the "EMACS" with a subroutine and a loop.

D. IIR Filter Implementation

The infinite impulse response (IIR) filter implementation requires more efforts. The scaling factors must be incorporated to avoid a possible overflow for each accumulator and a coefficient quantization overflow due to the designed coefficient value larger than one. An illustrative example is given below:

$$H(z) = \frac{0.06747 + 0.1349z^{-1} + 0.06747z^{-2}}{1 - 1.1429z^{-1} + 0.4128z^{-2}}$$

Students are first required to implement H(z) using direct form I shown in Figure 12.

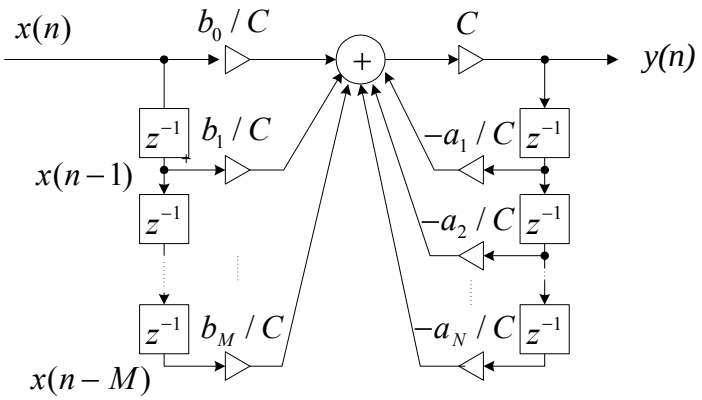


Figure 12. IIR filter implementation in direct form I.

Converting the transfer function yields the following difference equation:

$$y(n) = 0.06747x(n) + 0.1394x(n-1+0.06747x(n-2) + 1.1429y(n-1) - 0.4128y(n-2)$$

Since a coefficient value of 1.1429 is larger than 1, the DSP equation must be scaled down by a factor of 2 to avoid the coefficient quantization overflow. The scaled DSP equations are

$$y_s(n) = \frac{0.06747}{2}x(n) + \frac{0.1394}{2}x(n-1+\frac{0.06747}{2}x(n-2) + \frac{1.1429}{2}y(n-1) - \frac{0.4128}{2}y(n-2)$$

$$y(n) = 2y_s(n)$$

The quantized coefficients using 16 bit including a sign bit are listed below:

$$\frac{0.06747}{2}\times 2^{15}\approx 1106,$$
$$\frac{0.1344}{2}\times 2^{15}\approx 2210,$$
$$\frac{1.1429}{2}\times 2^{15}\approx 18725,$$
$$-\frac{0.4128}{2}\times 2^{15}\approx -6763$$

Figure 13 depicts the memory and buffer organization.

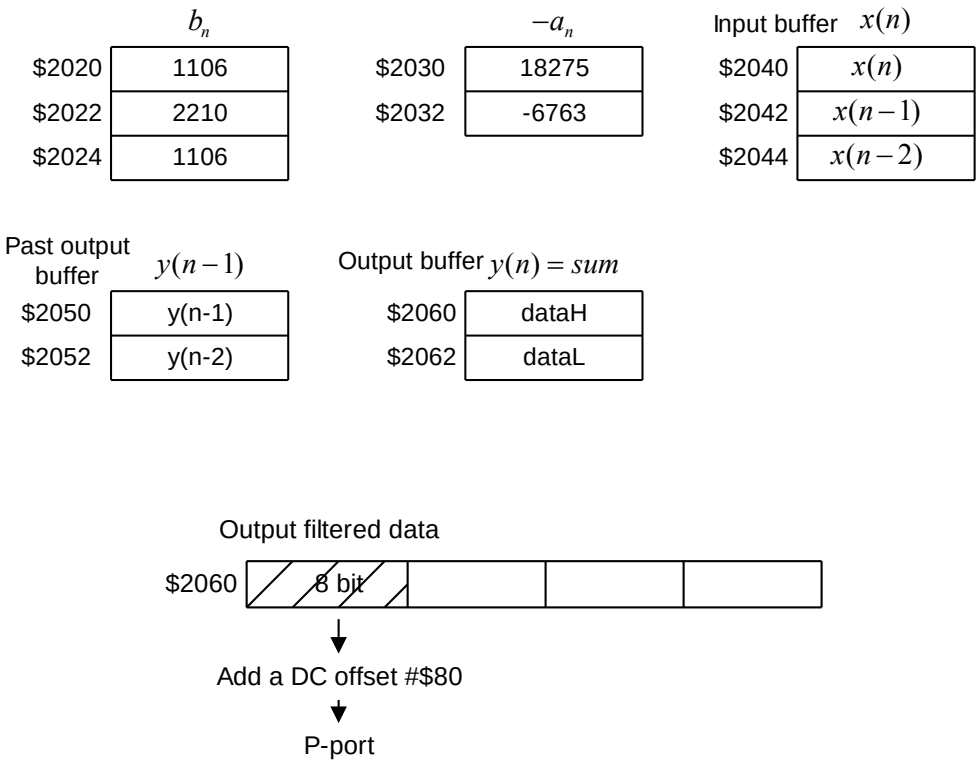


Figure 13. Memory arrangement for an IIR filter in direct form I.

Figure 14 illustrates the IIR filter implementation using linear buffers. Note that both input and output data buffers need to be updated for processing each input sample. Figure 15 shows a sample program segment.

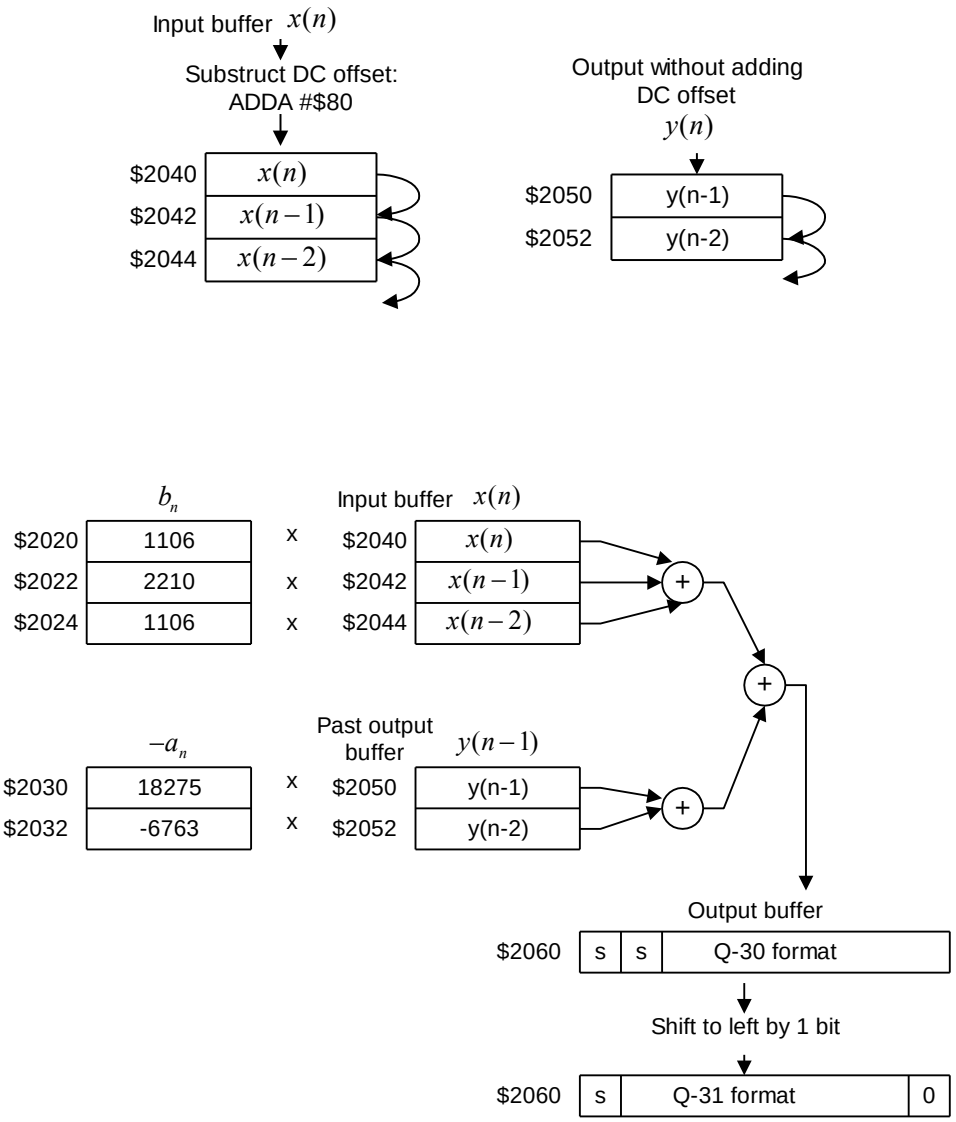


Figure 14. IIR filter implementation in direct form I.

```
 …      ; sampling rate control
LDD $2060        ;output filtered data
STAA $0056       ; send y(n) to P-Port
LDD #$00    ; clear accumulate
STD $2060
STD $2062
…        ; ADC
LDD $2042   ;update the linear buffer for input x(n)
STD $2044
LDD $2040
STD $2042
LDAA $0074  ;update sample
LDAB #$00  ;clear lower byte
ADDA #$80 ; subtract DC offset =2.5 volts
STD $2040
LDX #$2020  ; perform b0x(n)+b1x(n)+b2x(n-2)
LDY #$2040
EMACS $2060
INX
INX
INY
INY
EMACS $2060
INX
INX
INY
INY
EMACS $2060  ;done with sum of b(n)*x(n)
LDX #$2030 ; perform  -a1y(n-1)-a2y(n-2)
LDY #$2050
EMACS $2060
INX
INX
INY
INY
EMACS $2060
LDD $2060; change y(n) in Q-15 and scale it up by 2
LSLD
LSLD ; scale factor used in quantization
STD $2060
LDD $2050   ; update buffer y(n-1) y(n-2)
STD $2052
LDD $2060
STD $2050
LDAA $2060
ADDA #$80      ;add DC offset
STAA $2060
```

Figure 15. A sample program segment for IIR filter implementation in direct form I.

Next, direct form II shown in Figure 16 can be investigated using the same IIR filter transfer function.

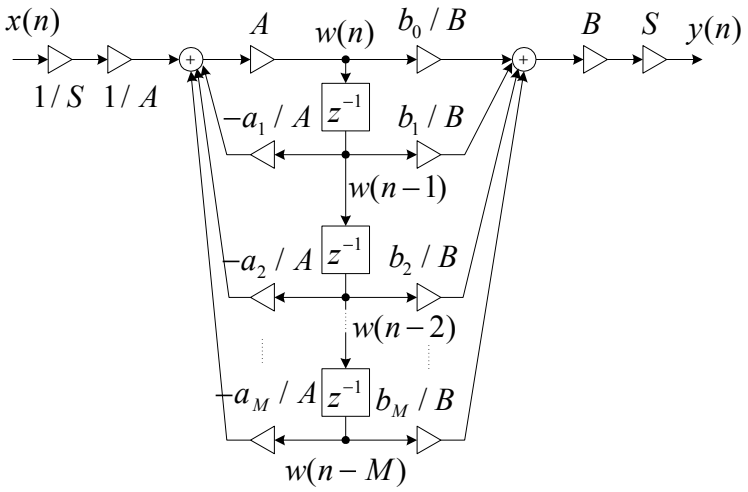


Figure 16. IIR filter implementation in direct form II.

Converting the transfer function yields the following difference equations in direct form II:

$$w(n) = x(n) + 1.1429w(n-1) - 0.4128w(n-2)$$
$$y(n) = 0.06747w(n) + 0.1349w(n-1) + 0.06747w(n-2)$$

To avoid the overflow in the first equation (corresponding to the first adder), an impulse response sequence from its transfer function can be determined by using MATLAB as follows:

$$h(n) = Z^{-1}\left(\frac{1}{1-1.1429z^{-1}+0.4128z^{-2}}\right)$$

The scale factor S shown in Figure 16 is then computed using the following formula[4]:

$$S = \sum |h(n)| \approx 4$$

For this case, S=4 is selected. Again, since the coefficient 1.1429 >1, a scale factor of A=2 is employed to ensure all the numerator coefficients in the first equation are less than 1. B=1 is chosen since all of the coefficients in the second equation are fractions. Thus, the final implementation equations without coefficient quantization are listed below:

$$x(n) = \frac{1}{4} x_{input}(n)$$
$$w_s(n) = \frac{1}{2} x(n) + \frac{1.1429}{2} w(n-1) - \frac{0.4128}{2} w(n-2)$$
$$w(n) = 2w_s(n)$$
$$y(n) = 0.06747w(n) + 0.1349w(n-1) + 0.0647w(n-2)$$
$$y_{output}(n) = 4y(n)$$

Now, quantizing each coefficient leads to

$$\frac{1}{2} \times 2^{15} = 16384,$$
$$\frac{1.1429}{2} \times 2^{15} \approx 18726,$$
$$-\frac{0.4128}{2} \times 2^{15} \approx -6764,$$
$$0.06747 \times 2^{15} \approx 2211,$$
$$0.1349 \times 2^{15} \approx 4420$$

The memory arrangement for input and output data buffers is displayed in Figure 17. As shown in Figure 17, there are three data buffers: input data buffer, output data buffer, and state data buffer. Only state data buffer requires the FIFO operation. A detailed implementation is shown in Figure 18.

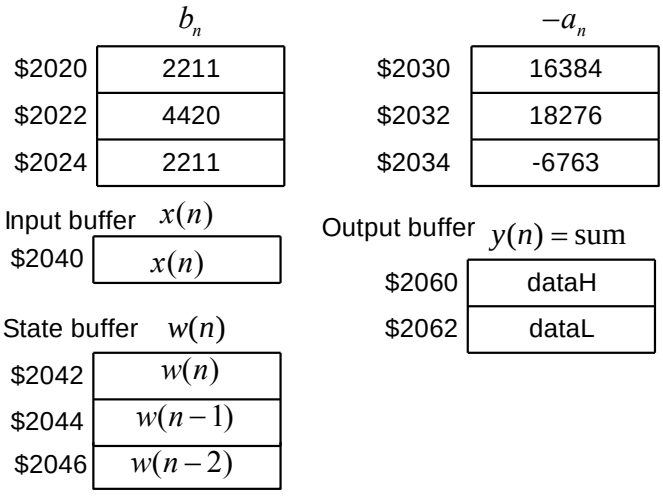


Figure 17. Memory arrangement for IIR filter implementation in direct form II.

```
…       ; sampling rate control
LDD $2060  ;output the filtered data
STAA $0056
LDD #$00
STD $2060
STD $2062
…       ; done with ADC
LDAB $0074  ;update sample
LDAA #$00
ADDD #$FF80 ;add DC offset with sign extension
LDY #$0040  ;scale the input x(n) down by 4 (S=4)to avoid overflow
emuls      ;note that A=2 is included in coefficients
STD $2040
LDX #$2030
LDY #$2040
EMACS $2060      ;0.5*x(n)
INX
INX
LDY #$2044
EMACS $2060       ;-a1*w(n-1)
INX
INX
INY
INY
EMACS $2060     ;-a2*w(n-2)
LDD $2060
LDY #$0002     ; A=2
emuls
LSLD          ; adjust to Q-15
STD $2042       ; w(n)
LDD #$0
STD $2060
STD $2062
LDX #$2020  ; filtering data
LDY #$2042
EMACS $2060     ;b0*w(n)
INX
INX
INY
INY
EMACS $2060    ;b1*w(n-1)
INX
INX
INY
INY
EMACS $2060  ;done with sum of b(n)*w(n)
LDD $2044   ;update the linear buffer for w(n)
STD $2046
LDD $2042
STD $2044
LDD $2060
LSLD  ; change y(n) to Q-15
LSLD  ; scale up by 4, S=4
LSLD
ADDA #$80    ;add DC offset
STD $2060
```

Figure 18. A sample program segment for IIR filter implementation in direct form II.

With the established knowledge and sample programs, students can further conduct their own filter implementations using their own designed filters. Finally, a group project can be assigned to students to develop more advanced implementations including dual tone multi-frequency (DTMF) tone generation using IIR filters, FIR filter using the circular buffering, and sampling rate conversions.

III. Student Evaluation and Improvement

Upon completion of the DSP course as well as its laboratory experiments, a survey was conducted to ask each student to evaluate his/her achievement using the 68HC12 microcontroller as a learning tool. Table 1 shows the survey results. Note that the rating scale was based on the percentage of the overall students.

Table 1. Student survey for their achievements.

Rating scale	Understanding of digital filter implementation	Tools	Excitement
4 – excellent	85%	90%	80%
3 – good	15%	10%	15%
2 – fair	0%	0%	5%
1 – unsatisfied	0%	0%	0%

Most students remained excited about labs since the hands-on real-time labs using their familiar platform motivated them. Students felt that they can focus on learning filter implementations without putting extra effort to learn the new programming tool and environment. The textbook[4] helped a lot to develop DSP concepts using ample worked numerical examples accompanying with handy MATLAB simulation examples and programs. After learning the digital filter implementation, students enhanced their skills in the embedded system design significantly so that they could apply their gained knowledge and proficiency into their senior capstone projects. Our future improvement could include developing more practical projects with applications of processing low frequency signals like instrumentation, vibration, and biomedical signals.

IV. Conclusion

In this paper, we have demonstrated the feasibility and our pedagogy for teaching a real-time DSP course using the 68HC12 microcontroller. We have validated that using the 68HC12 microcontroller as a platform in our DSP course is not only cost-effective but also learning effective. The developed method could be an alternative when the DSP dedicated hardware is not available while offering a DSP course is in demand.

Bibliography

1. D. J. Pack, S. F. Barrett, 68HC12 Microcontroller: Theory and Applications. Prentice Hall, 2002.
2. Axiom Manufacturing: http://www.axman.com/

3. D. Grover, J. R. Deller, Digital Signal Processing and the Microcontroller. Prentice Hall, 1999.
4. L. Tan, Digital Signal Processing: Fundamentals and Applications. Elsevier/Academics, 2007.
5. T. B. Welch, C. H.G. Wright, M. G. Morrow, Real-Time Digital Signal Processing. CRC Press, 2006.