Int. J. Engng Ed. Vol. 23, No. 5, pp. 1000–1010, 2007
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Low-cost Microcontroller-based Hardware for Introducing Digital Filter Fundamentals to Students*

DOGAN IBRAHIM
Department of Computer Engineering, Near East University, Nicosia, Cyprus
E-mail: dogan@neu.edu.tr

Digital filters are currently created using high-performance DSP chips. Although these chips have many advantages in commercial and industrial applications, they usually oversimplify and hide away the basic concepts for the realization of digital signal processing techniques. This paper describes the design of a digital filter using a low-cost microcontroller as the processing element. The main feature of the system is that it aims to teach students the basic hardware and software implementation of digital filters. Both FIR and IIR type filters can easily be implemented with the system; the developed platform will be used at the Near East University together with DSP chips to teach the practical realization of digital filters to undergraduate students.

Keywords: Digital filter; FIR filter; IIR filter; DSP; microcontroller, MATLAB

INTRODUCTION

IN SIGNAL PROCESSING signals are often encountered that contain unwanted information, such as random noise or interference, or it may be desired to extract a signal of interest from a collection of a number of signals. Filters are used in such applications to separate the desired signal from the unwanted signals.

Filters can be analogue or digital. Analogue filters [1] can be passive (e.g. resistor, capacitor, inductor) or active (e.g. operational amplifiers and passive components) and they use electronic components to separate the desired frequency. Analogue filter theory is well established and there are many books and papers that cover the design and implementation of such filters, but the theory is beyond the scope of this paper.

Digital filters [2, 3, 4] use a digital processor to implement an algorithm to perform numerical calculations on sampled values of the signal. The processor in a digital filter can be a general purpose microprocessor, such as the Z80, 8086 series etc., or a microcontroller such as the PIC series, 8051, 6805 etc., or a specialized Digital Signal Processor (DSP) chip such as the TMS320 series or ADSP 210, or an Application Specific Integrated Circuit (ASIC) incorporating a DSP core processor such as the TEC320C52 or D950-CORE.

Recently, digital filters have been implemented using high-performance, expensive DSP chips [5]. These are optimized to process large amounts of real-time data efficiently, making them ideal for numerically intensive DSP applications, such as FFTs, digital image processing, high-fidelity audio processing and digital filters. There are dozens of DSP chips on the market with complex architectures, some capable of performing 16 bit fixed-point arithmetic, such as the TMC320C54x, DSP561xx or the ADSP-21xx series. Some chips, such as the DSP563xx or the DSP5600x offer 24 bit fixed-point arithmetic capability. Other more complex chips operate with 32 bit floating-point arithmetic such as the ADSP-210xx, DSP96002 or the TMS320C4x. In fixed-point processors, numbers are represented as integers or in fractional format. Here, the designer has to convert the filter coefficients into fractional numbers and take care to scale the signals at various stages of the algorithm to avoid any overflows in the calculations. Most high-volume embedded applications where low cost is a prime factor use fixed-point processors, although floating-point processors provide a greater flexibility and ease of design. In addition, with floating-point processors the system designer has access to a wider dynamic range (the ratio between the largest and the smallest number that can be represented) and also greater precision. In real applications, dynamic range is very important as it translates into signal magnitudes that can be processed while maintaining sufficient fidelity in the signal. Some applications, such as communications, require moderate dynamic ranges, but applications such as high-fidelity audio applications require higher dynamic ranges. Most applications that require high precision and very high dynamic range and where the cost is of no concern use floating-point processors.

Manufacturers of DSP chips usually offer general purpose software and hardware DSP

* Accepted 5 May 2007

development kits. Some kits are complete development systems with a board having a DSP chip and associated interface circuitry, a complete set of software including a compiler and library files for the development of algorithms, and simulators to test the algorithm before it is downloaded to the chip. Some researchers have extended the capabilities of DSP kits by providing more input and output ports. Alonso and Barreto [6] describe a practical, economical and useful way of enhancing the Texas Instruments TMS320C3x DSP learning kit. All DSP vendors provide the basic assembly language software and assembler tools to develop the algorithm in assembly code. High-level language compilers such as C and application libraries are also offered for those who wish to program using a high-level language. Such programming has the advantages of considerably reduced development and testing times.

DSP chips are well suited to commercial and industrial high-speed signal processing applications as they remove the need to construct complicated, expensive and specialized hardware. A DSP development kit usually contains all the required hardware for the implementation of a signal processing application, additionally provides development software that makes the design and testing a relatively simple task. Although students can use DSP development kits to implement and test various signal processing algorithms, this does have some disadvantages at the introductory learning stage, namely:

- DSP kits are usually expensive.
- DSP kits hide the basic structure and parts of a signal processing hardware from the students. Students using such development kits are not usually aware that a digital signal processor can be designed from first principles using an A/D converter, a general purpose processor and a D/A converter.
- DSP kits have complex structures and it may take a considerable amount of time to learn their architectures.
- Using a DSP development kit students learn how to handle the products of a particular manufacturer only. Different manufacturers offer different and usually incompatible kits which add to the existing learning curve.

The hardware developed at the Near East University using the basic electronic building blocks is used to teach the design and implementation of digital filters to third year undergraduate students studying at the Department of Computer Engineering. The idea behind this research work was to develop a signal processing hardware from first principles that will teach students the inside of signal processing hardware and software. They calculate filter coefficients using a MATLAB program, then use these coefficients to realize a practical digital filter using an A/D converter, a low-cost microcontroller and a D/A converter. It should be realized that this approach is not for practical application, but rather to teach a deeper insight into the process of digital filtering. After learning the basic theory and practical realization of digital filters students move on to experiment with complex DSP development kits. The theory of digital filters is well established and there are numerous articles and books on this topic. Gan and Kuo [7] propose a two-level approach for teaching digital signal processing where MATLAB and Simulink [8, 9] are used to make the transition from theory to application. They propose that a two-stage approach eases the learning of theory and also makes the learning process more enjoyable.

Undergraduate engineering courses in all universities teach the basic theory and applications of various microcontrollers and the C language. The hardware and software approach provided in this design is an extension of what most students already know, i.e. it combines the real-time implementation of a microcontroller- based digital filter and the familiar C language.

REQUIREMENTS

Near East University was established in 1988 in Cyprus, and currently has over 13,000 students from all over the world, studying in various departments in 14 faculties. The Department of Computer Engineering offers full-time undergraduate and postgraduate courses in the faculty of engineering. Students on the undergraduate course learn all aspects of computer hardware and software, including basic electronics, logic design, the theory and applications of microprocessors and microcontrollers and various programming languages and applications. Students in the final year are offered various options, signal processing being one of them. In the past, students were taught MATLAB programming; then they were asked to use a PC to calculate digital filter parameters and simulate them in the MATLAB environment. Although simulation can be an invaluable tool in engineering teaching, it cannot replace the experiments carried out in a real laboratory environment. This was the main requirement for developing the hardware.

DIGITAL FILTER HARDWARE

We have used a low-cost PIC microcontroller [10] as the processing element. The PIC family is currently very popular in industry and in education and many students are familiar with the architecture and programming of these chips. PIC microcontrollers are manufactured by Microchip Inc. There are over a hundred in the family; most of them are compatible with each other; some advanced members include on-chip functions such as A/D converters, USART, I^2C bus interface, SPI bus interface, USB interface and so on. The micro-

controller we used was the PIC16F877, a 40-pin processor with following specifications. Other lower specification members of the PIC family such as the PIC16F870 could also have been used:

- 8192 byte program memory;
- 368 byte RAM memory;
- 256 byte EEPROM memory;
- 33 parallel I/O ports;
- 14 interrupt sources;
- 8-channel multiplexed 10-bit A/D converters;
- USART;
- I^2C and SPI bus compatible interface.

The device can operate with a clock rate up to 20MHz; a 20MHz crystal was used as the clock source. At this rate the basic instruction execution cycle is 0.2μs.

Figure 1 below depicts a block diagram of the designed digital filter hardware. It is important to realize that although the microcontroller used has built-in A/D converter ports these were not used, but an external A/D converter chip was incorporated into the design. The reasons were first, a bipolar converter was required in the design and the on-chip converter supported unipolar voltages only. Second, because it aimed to teach the basic signal processing hardware building blocks. The A/D converter receives the analogue signal to be filtered and converts this into digital form. The A/D converter operates under control of the microcontroller. The converted digital signal consists of a sequence of numbers which are fed to the microcontroller for processing. The microcontroller implements the digital filter algorithm and sends the filtered digital signal to the D/A converter. The D/A converter produces the required filtered analogue output. It was necessary to use a bipolar D/A converter in this project since signals are normally bipolar.

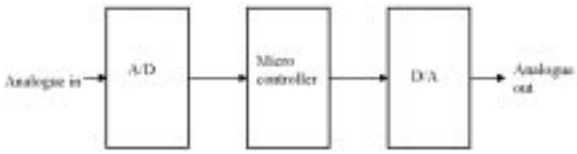

Fig. 1. Block diagram of the digital filter hardware

The circuit diagram of the digital filter is shown in Figure 2.

The A/D Converter

In this project an 8-bit A/D converter was used for simplicity. The input analogue signal voltage range was ±5V and the converter provided a quantized digital signal with 256 steps. The converter used was the AD673, manufactured by Analog Devices Inc [11]. This converter has a 20μs conversion time, accepts both unipolar and bipolar input signals and operates with two power supplies, +5V and −12V. Basically the device includes:

- An analogue input pin;
- 8 digital output pins;
- A CONVERT input pin;
- A DATA READY output pin.

Conversion starts by applying the analogue signal

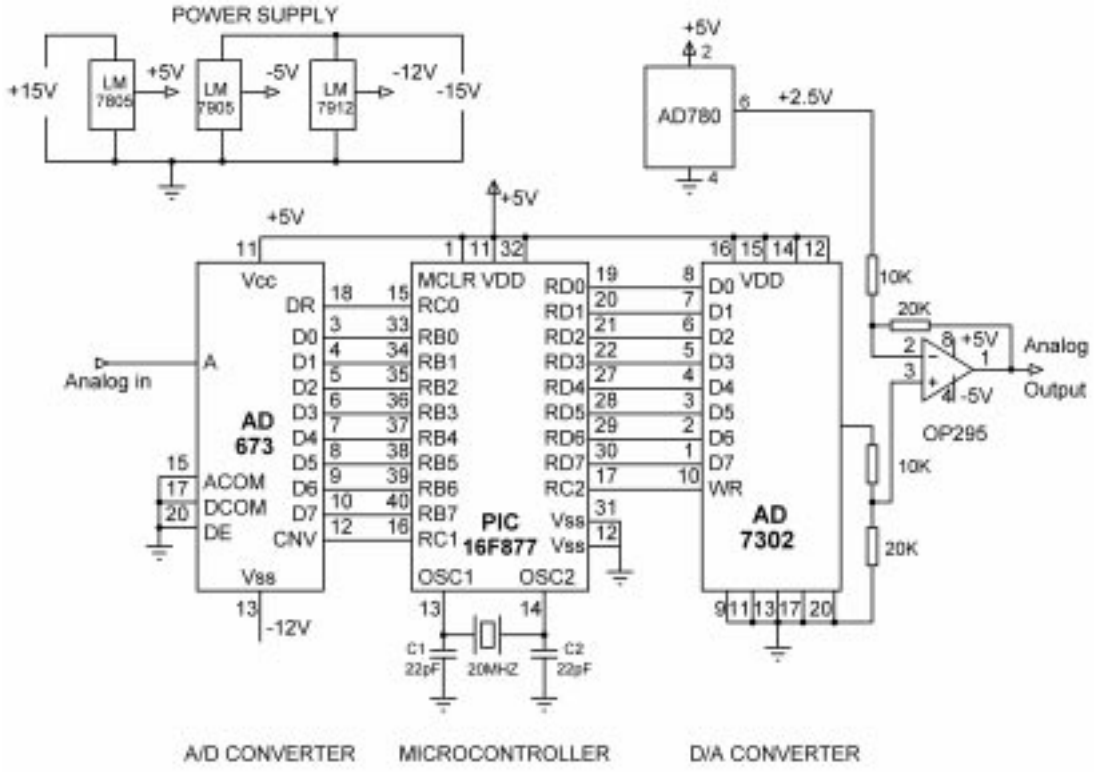

Fig. 2. Circuit diagram of the digital filter

Table 1. Some offset binary numbers and their 2's complement equivalents

Scale	Offset Binary	2's Complement
+ Full scale	11111111	01111111
+0.75 Full scale	11100000	01100000
+0.50 Full scale	11000000	01000000
+0.25 Full scale	10100000	00100000
0	10000000	00000000
−0.25 Full scale	01100000	11100000
−0.50 Full scale	01000000	11000000
−0.75 Full scale	00100000	10100000
− Full scale	00000000	10000000

to the input pin and then pulsing the CONVERT pin. The high-to-low transition of the CONVERT pulse starts the conversion and the DATA READY pin goes to logic high. After about 20μs the conversion is complete and data are available at the digital outputs. The A/D converter completion is signalled by the DATA READY pin going low. This output is used to inform the microcontroller that the conversion is complete and the converted data are available at its output pins.

AD673 A/D converter provides offset binary signals at its outputs. Offset binary is similar to 2's complement notation but the MSB bit is inverted. Some 8-bit offset binary numbers and their 2's complement equivalents are given in Table 1.

2's complement is the commonly used numbering system in signal processing applications. Numbers represented in 2's complement notation can easily be added, subtracted, multiplied or divided. A given offset binary number can easily be converted to 2's complement form by simply inverting the MSB bit.

The D/A converter

In this project an AD7302 type D/A converter was used. This device is a dual channel D/A converter, manufactured by Analog Devices Inc. Digital data are sent to the converter and, under the control of the microcontroller, the converter produces analogue voltage at its output. Basically the device includes:

- 8 digital inputs;
- Analogue output (two channels);
- WR write input;
- Channel select input;
- Reference input.

Conversion starts by sending digital data to the converter and pulsing the WR input. After about 2μs the conversion is complete and the analogue data are available at the output of the converter. As in the A/D converter, the D/A converter operates with offset binary and the input signal to the converter must be in this format. Thus, it is necessary to complement the MSB bit of the data before sending it to the D/A converter. Normally, the output of the AD7302 is unipolar and an operational amplifier was used to produce bipolar signals in the range ±5V. The operational amplifier used in the project was the OP295 which is a quad operational amplifier package manufactured by Analog Devices Inc. One of the reasons for choosing this operational amplifier was because it provides a rail-to-rail output voltage, and also it has a high output drive capability.

As shown in Figure 2 above, PORT B of the microcontroller was connected to the output of the A/D converter, and PORT D drives the D/A converter. Port pin RC0 and RC1 are connected to DATA READY and CONVERT inputs of the A/D converter respectively. Port pin RC2 of the microcontroller controls the WR input of the D/A converter. Figure 3 below shows the I/O port mapping of the microcontroller. Power is provided to the circuit using a pair of LM7805 and LM7905 voltage regulator chips for the -5V and +5V rails respectively, and a LM7912 chip for the +12V rail.

DIGITAL FILTER ALGORITHMS

A digital filter, in its most general form, takes in an input sequence of numbers x[n], performs computations on these numbers and outputs result as another sequence of numbers y[n]. The relationship between the input and output is given by:

$$y(nT) = \sum_{k=0}^{M} a_k x(nT - kT) - \sum_{k=1}^{N} b_k y(nT - kT) \quad (1)$$

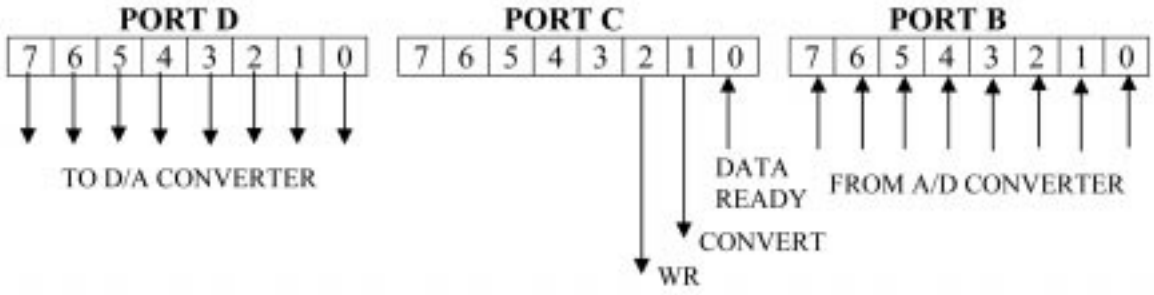

Fig. 3. I/O port mapping of the microcontroller

the above equation has the z-transform

$$H(z) = \frac{y(z)}{x(z)} = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}} \quad (2)$$

There are two types of digital filters: infinite impulse response (or IIR) filters and finite impulse response (or FIR) filters [12–15]. Equation (1) is the transfer function of an IIR type filter where the output samples depend on the present input and past output samples. In this equation, if the output samples depend only on the past input samples, i.e. $b_k = 0$ for $k = 1, 2.....M$ then the realized filter is known a FIR filter and its transfer function reduces to:

$$H(z) = \frac{y(z)}{x(z)} = \sum_{k=0}^{M} a_k z^{-k} \quad (3)$$

Design of a FIR digital filter requires calculation of the coefficients a_k. Similarly, design of an IIR filter requires the calculation of coefficients a_k and b_k. There are many references, text-books, computer programs and online active internet tools for the calculation of filter coefficients for a required filter specification.

A MATLAB-based program has been developed [16, 17] at the Near East University [18] to calculate the filter coefficients and then plot the theoretical response of the resulting filter. The program can be used to design both IIR and FIR type filters. For the FIR filters, the cut-off frequency, filter length and the windowing technique to be used are specified. The program calculates and displays the filter coefficients. Additionally, the frequency and responses of the filter are plotted. For the IIR filters the program can be used to design Butterworth, Chebyshev and Elliptic type filters. The cut-off frequency, filter order and required pass-band ripple are entered; the program calculates and displays the numerator and denominator coefficients and also plots the frequency and the phase responses of the filter.

Figure 4 shows the output produced when a second order low-pass Butterworth IIR filter with a cut-off frequency of 100 Hz and a sampling frequency of 1 kHz is designed using the program. Note that the filter coefficients are displayed in two list-boxes on the right-hand side of the display.

IMPLEMENTATION OF A LOW-PASS IIR FILTER

This section shows how an IIR filter can be designed and implemented using the hardware developed. A second-order Butterworth IIR type low-pass digital filter was designed to test the hardware. The filter had the following specification:

- 2nd-order Butterworth low-pass IIR filter;
- Sampling frequency = 1 kHz;
- Cut-off frequency = 100 Hz.

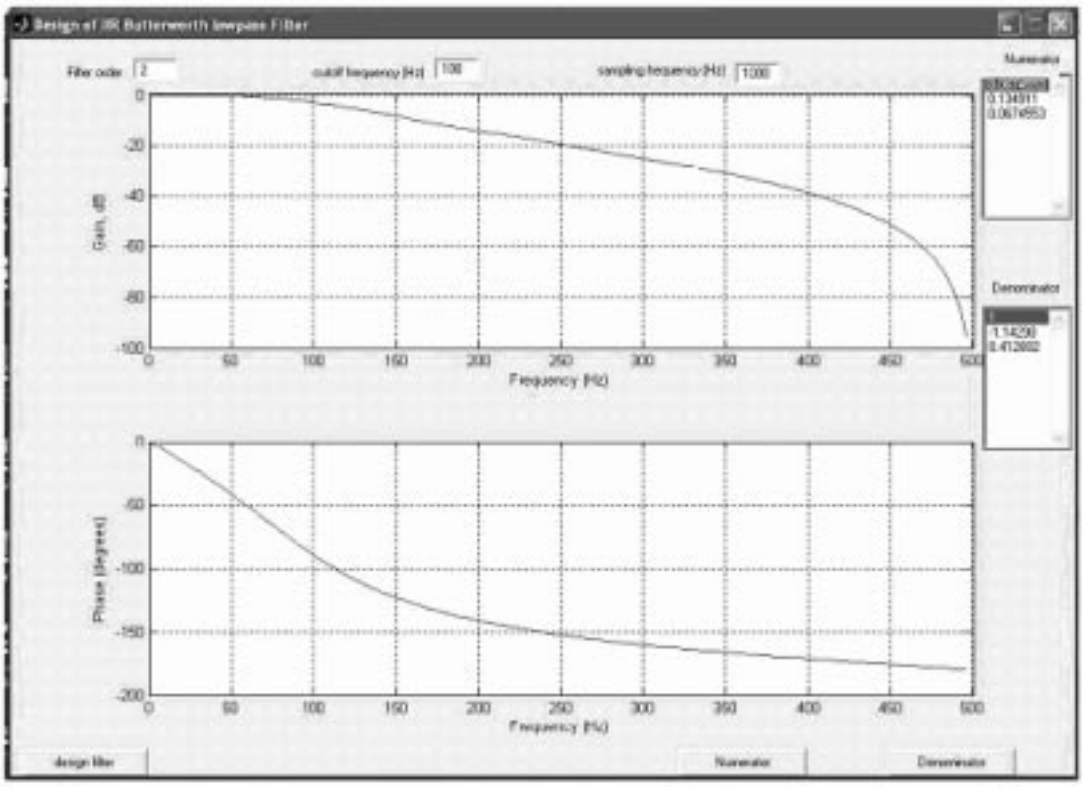

Fig. 4. Designing a 2nd-order IIR Butterworth low-pass filter

The filter coefficients were obtained running the MATLAB program discussed earlier. These coefficients were:

$b_0 = 0.06745$ $b_1 = 0.13491$ $b_2 = 0.06745$
$a_0 = 1$ $a_1 = -1.14298$ $a_2 = 0.412801$

Based on these coefficients, the transfer function of the required filter is:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} + a_2 z^{-2}} \tag{4}$$

or,

$$H(z) = \frac{0.06745 + 0.13491 z^{-1} + 0.06745 z^{-2}}{1 - 1.14298 z^{-1} + 0.412801 z^{-2}} \tag{5}$$

which can be written as

$$H(z) = \frac{0.06745(1 + 2z^{-1} + z^{-2})}{1 - 1.14298 z^{-1} + 0.412801 z^{-2}} \tag{6}$$

writing the above transfer function as

$$H(z) = \frac{y(z)}{u(z)} = \frac{K(1 + 2z^{-1} + z^{-2})}{1 + Bz^{-2} + Cz^{-2}} \tag{7}$$

where

$$K = 0.06745 \quad B = -1.14298 \quad C = 0.412801$$

Let,

$$\frac{y(z)}{u(z)} = \frac{y(z)}{q(z)} \frac{q(z)}{u(z)} \tag{8}$$

where,

$$\frac{q(z)}{u(z)} = \frac{K}{1 + Bz^{-1} + Cz^{-2}} \tag{9}$$

and

$$\frac{y(z)}{q(z)} = 1 + 2z^{-1} + z^{-2} \tag{10}$$

from (9),

$$q(z) = Ku(z) - Bq(z)z^{-1} - Cq(z)z^{-2} \tag{11}$$

and from (10),

$$y(z) = q(z) + 2q(z)z^{-2} + q(z)z^{-2} \tag{12}$$

Let x_1 and x_2 be two state variables, where $x_1 = q(z)z^{-1}$and $x_2 = q(z)z^{-2} = x_1 z^{-1}$

Then we can write,

$$q(z) = Ku)(z) - Bx_1 - Cx_2 \tag{13}$$

$$y(z) = q(z) + 2x_1 + x_2 \tag{14}$$

Thus, the following operations will be required to implement the second-order filter section:

$$Ku - Bx_1 - Cx_2 \rightarrow q$$
$$q + 2x_1 + x_2 \quad \rightarrow y$$
$$x_1 \rightarrow x_2$$
$$q \rightarrow x_1$$

The block diagram of the second-order filter implementation is shown in Figure 5. This implementation is also known as the direct-form of realization, or the bi-quad section.

The steps required for the implementation of a second-order filter are summarized below:

- Enable timer interrupt;
- Input u from A/D converter;
- Calculate $Ku - Bx_1 - Cx_2$;
- Store result in q;

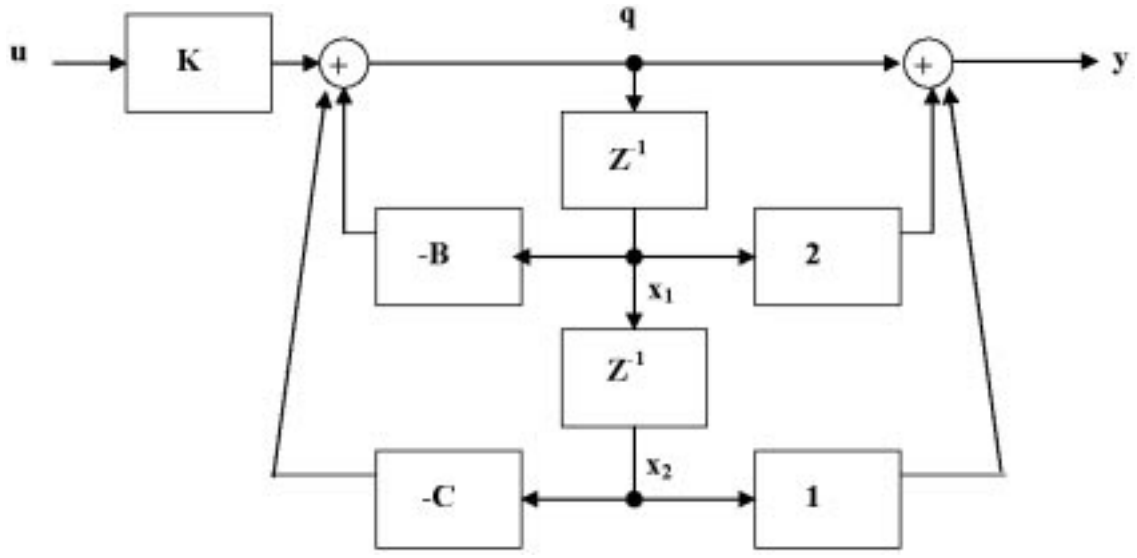

Fig. 5. 2nd-order IIR filter implementation

```
                    2nd-order Butterworth Low-Pass IIR Digital Filter
                    = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
This is the program for a 2nd-order Butterworth IIR digital filter. The filter is implemented on a PIC16F877
microcontroller, operating At 20MHz clock. The basic instruction cycle time is 0.2 microsecond.
The filter parameters are as follows:
• Sampling frequency = 1kHz;
• Cut-off frequency = 100 Hz.

The filter is implemented in a timer interrupt routine (ISR). The ISR is called every 1000 microsecond (1kHz)
and the filter algorithm is implemented. The steps at every 1000 microseconds are basically:
• Read analogue input;
• Process data;
• Send out analogue output.

File:FILTER.C
Date:    February 2006
********************************************************************/
      #include <pic.h>
      #include <delay.c>
      // Declare variables
      float x1,x2,q,fu,temp,K,B,C;
      signed char u,y,j;
      // Declare symbols
      #define     DATA_READY        RC0
      #define     CONVERT           RC1
      #define     WR                RC2

/****************** INTERRUPT SERVICE ROUTINE ********************
This routine is called at every 1000 microseconds. Inside the routine the analog  data is received from the A/D
converter, the digital filter algorithm is implemented and the data is sent to the D/A converter */

void interrupt filter(void)
{
      TMR0 = 100;                       // Re-load TMR0
      // Start A/D conversion
      CONVERT = 1;
      DelayUs(2);
      CONVERT = 0;
      while(DATA_READY);                        // Wait until DATA_READY= 0
      // Read converted analogue data
      u = PORTB;                        // Get converted data
      u = u ^ 0x80;                     // Convert to 2s complement
      fu = (float)u;                    // A/D data in floating point, fu
      // Implement the filter algorithm
      q= K*fu-B*x1-C*x2;
      temp = q+x1+x1+x2;
      // Get output sample
      y = (signed char)temp;
      // Convert to offset binary
      y = y ^ 0x80;
      // Send to D/A converter
      PORTD = y;
      WR = 0;
      WR = 1;                                   // Activaet D/A converter
      x2=x1;                            // Exchange filter states
      x1=q;
      // Re-enable TMR0 interrupts
      T0IF = 0;
}
/* ————————- Start of MAIN program ————————- */
main()
{
      // Initialize states and filter parameters
      x1 = 0.0;                  x2 = 0.0;
      K = 0.06745;     B = -1.14298;     C = 0.412801;
      // Initialize registers
      TRISB = 0xFF;                             // PORT B is all input
      TRISC = 1;                        // RC0 is input
      TRISD = 0;                        // PORT D is all output
      CONVERT = 0;                              // A/D in idle mode
      WR = 1;                                   // D/A in idle mode
      T0CS = 0; PSA = 0;                // Select TMR0
      PS0 = 0; PS1 = 0; PS2 = 1;        // Set timer pre-scaler to 32
      TMR0 = 100;                       // Set for 1000 us interrupt
      T0IE = 1;                         // Enable TMR0 interrupts
      T0IF =0;                                  // Enable TMR0 interrupt flag
      ei();                             // Enable global interrupts
WAIT: goto WAIT;                                // Wait for timer TMR0 interrupt
}
```

Fig. 6. Program listing of the digital filter

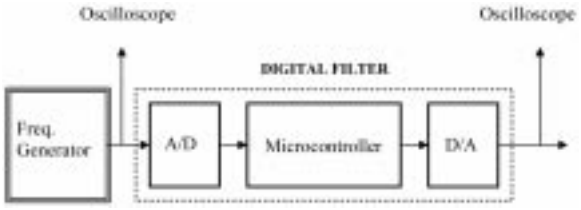

Fig. 7. Experimental setup

- Calculate $q + 2x_1 + x_2$;
- Output result to D/A converter;
- Perform $x_1 \rightarrow x_2$;
- Perform $q \rightarrow x_1$;
- Wait for timer interrupt;
- Repeat.

Higher-order filters can easily be implemented by cascading the basic second-order blocks.

The software

The PIC microcontroller was programmed using the C language. The compiler used was the PICC Lite, manufactured by Hi-Tech [19] and distributed free as a limited functionality compiler. Using a high-level language such as C has the advantage that algorithms can be developed and tested quickly and easily by students. In fact, students should be able to develop and implement the complete filter algorithm in a three-hour laboratory session. Another advantage of using a high-level language is that the program code can be transferred to other microcontrollers with usually minor or no modifications.

Filter coefficients and filter states are declared as floating-point variables, which makes the programming easier and also gives a greater dynamic range to the filter. The program is interrupt driven where timer interrupts establish the sampling time of the filter accurately. The timer TMR0 of the PIC18F877 microcontroller was programmed to generate an interrupt at every sampling time, i.e. at every 1000μs (sampling frequency = 1 kHz). The actual filter algorithm is then implemented inside the interrupt service routine. The program normally waits for a timer interrupt to occur; inside the interrupt service routine the A/D sample is read, the filter algorithm is implemented and the output signal is loaded to the D/A converter to produce analogue output from the filter. The operation of the software is described below:

- Initialize the program;
- Perform digital filtering continuously.

The initialization consists of:

- Initialize program variables and load filter coefficients;
- Initialize filter states to zero;
- Initialize port directions;
- Load TMR0 timer register for 1000μs interrupts;
- Enable timer interrupt;
- Wait for timer interrupts.

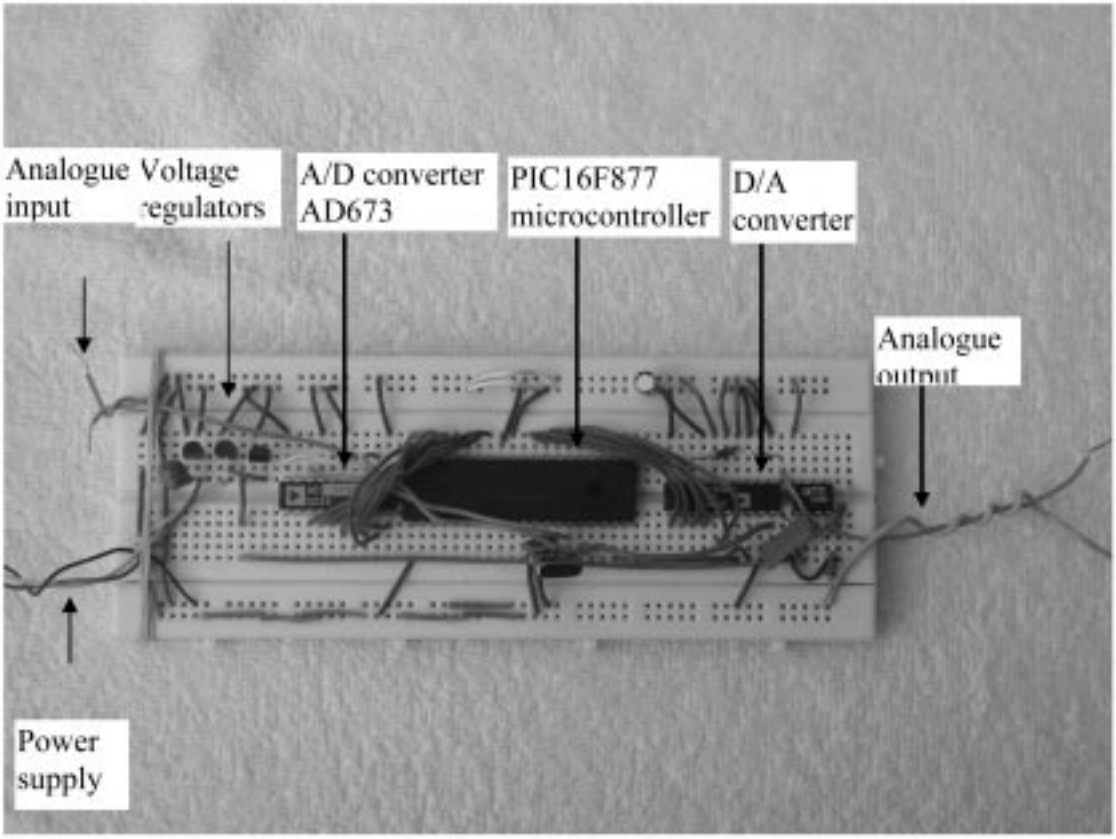

Fig. 8. Filter hardware

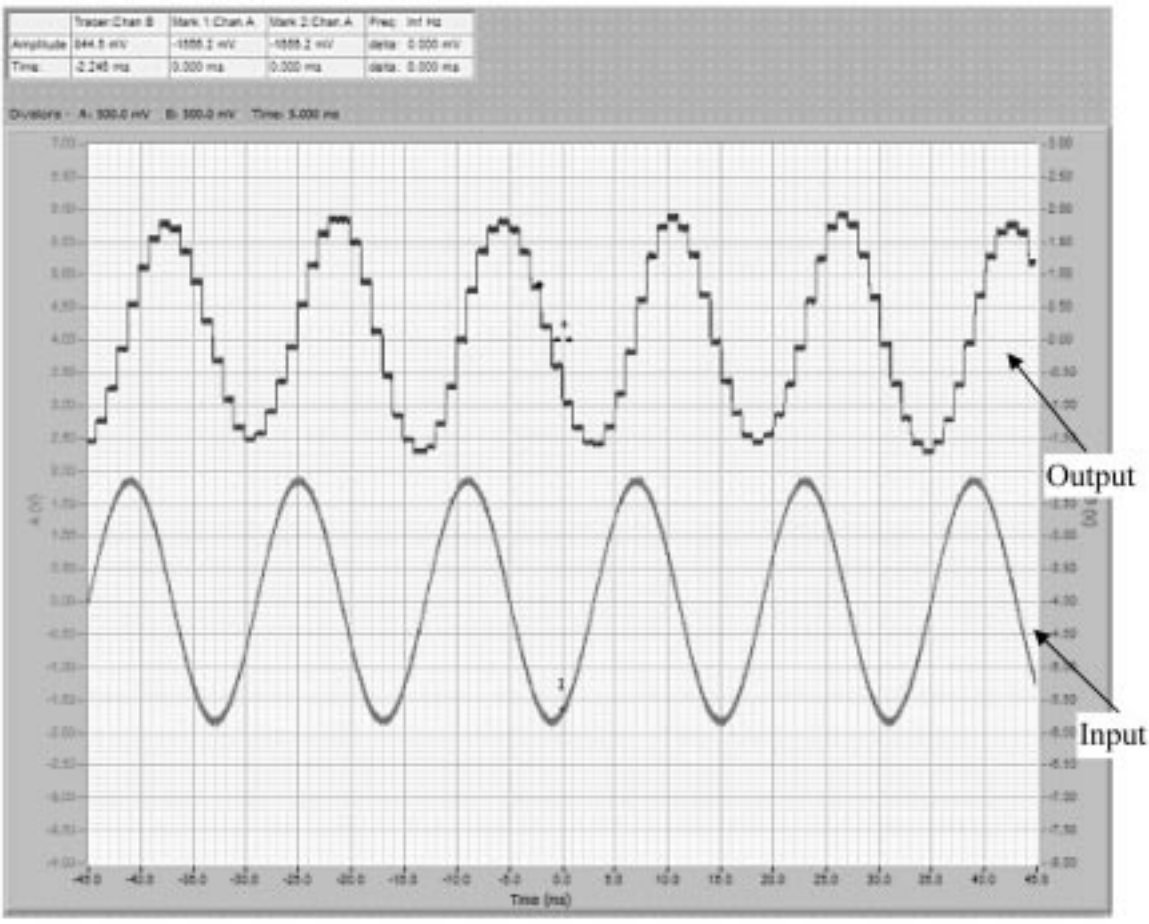

Fig. 9. Input and output waveforms of the digital filter

The digital filtering section of the program is entirely implemented in an interrupt service routine (ISR) so that the required sampling frequency is obtained accurately. The ISR consists of the following simple operations:

- Reload TMR0 timer register;
- Read a sample from A/D converter;
- Perform filtering algorithm;
- Send output to D/A converter;
- Return from interrupt.

On return from the ISR routine, the program goes back to the main program and waits for the next interrupt to occur. This way, the filtering algorithm is guaranteed to occur at every sampling interval. i.e. at every 1000μs.

The program listing of the filtering algorithm is given in Figure 6.

Experimental setup

Figure 7 shows a block diagram of the experimental setup. During the experiment, a PC-based two-channel oscilloscope, Cleverscope [20], is used by students to observe the input and output waveforms of the filter. The Cleverscope has a built-in audio frequency generator with an automatic sweep action and this is connected to the input of the filter. The filter was built on a breadboard and is depicted in Figure 8.

Figure 9 shows the typical input and output signal waveforms obtained from the filter and captured on the oscilloscope. The frequency response of the filter can be plotted using the built-in frequency spectrum analyser and the frequency sweep generator functions of the Cleverscope and this is depicted in Figure 10.

Students are asked to calculate the parameters of digital filters with given cut-off frequencies, then they implement these filters using the hardware proposed. Student success is then assessed by asking them to prepare a laboratory report giving the theory they have learned, also screen shots of the frequency response and filter input-output waveforms.

CONCLUSION

A microcontroller-based low-cost digital filter hardware platform has been described. The system consists of an A/D converter, a PIC microcontroller and a D/A converter. The filter algorithm has been implemented in C language using floating-point arithmetic which has simplified the program-

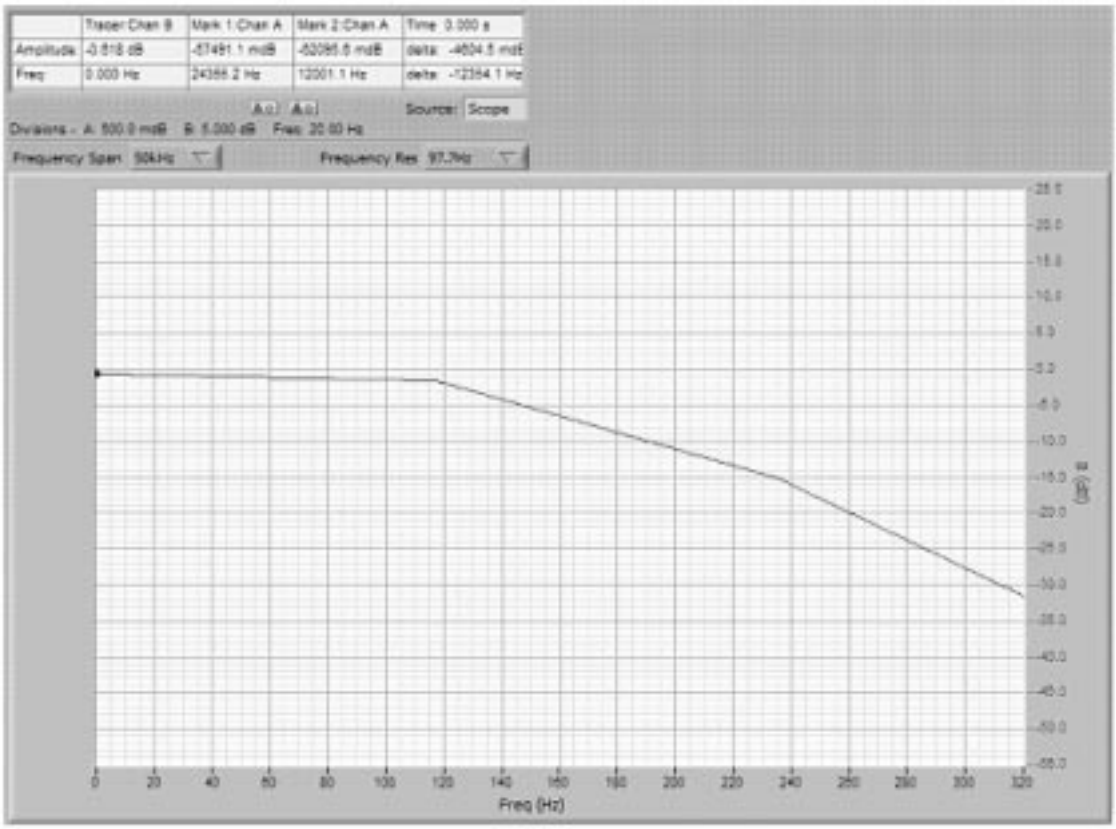

Fig. 10. Filter frequency response

ming. Although the actual instruction timing of floating-point operations has not been given by the developers of the C compiler, it is estimated that operation at up to about 10 kHz should be possible with a PIC microcontroller having a 20MHz clock. The cost of the developed hardware was not more than $50 (approx. £25). It was possible to lower the cost even further by using the built-in A/D converter of the microcontroller. It was, however, aimed to keep the basic DSP building blocks as distinct and separate as possible, so an external A/D chip was used instead. Another reason for using an external A/D converter was the need for bi-polar voltage levels.

Digital processing is traditionally implemented in industrial and commercial applications using DSP chips. Although this is useful, it can hide many of the basic concepts of signal processing from the students. The system designed and explained in this paper enables students to develop various signal processing algorithms and to implement and test them on the actual hardware. The hardware is low-cost, easy to learn, easy to construct and has the benefit that it teaches the principles of signal processing using the basic building blocks. It is intended to use the developed hardware initially, and then introduce the commercially available DSP kits for more complex signal processing experiments.

The filter coefficients were obtained by developing a MATLAB-based program where the user enters the filter specifications and the program displays the required filter coefficients, and also plots the theoretical frequency response of the filter so that students can compare the theory with practical results.

Although only the design of IIR type filters were considered in this paper, the same hardware can easily be used for the implementation of both FIR and IIR type filters.

REFERENCES

1. S. S. Soliman and M.D. Srinath, *Continuous and discrete signals and systems*, 2nd Ed., Prentice-Hall, Englewood Cliffs, (1998).
2. A.V. Oppenheim and R.W. Schafer, *Discrete-time signal processing*, Prentice-Hall, Englewood Cliffs, (1989).
3. S. J. Orfanidis, *Introduction to signal processing*, Prentice-Hall, Englewood Cliffs, (1996).
4. A. Bateman and W. Yates, *Digital signal processing design*, Computer science press, (1989).
5. P. Lapsley, J. Bier, A. Shoham and E.A. Lee, *DSP processor fundamentals*, IEEE press, (1997).
6. M. Alonso and A. Barreto, An Affordable Platform for Learning Real-Time Adaptive Signal Processing, *Int. J. Eng. Educ.* **20**(1) 2004, pp. 39–45.

7. S. M. Kuo and W. S. Gan, Transition from Simulink to MATLAB in Real-Time Digital Signal Processing Education, *Int. J. Eng. Educ*. **21**(4) 2005, pp. 587–595.
8. MATLAB *User's Guide*, Math Works, (2006).
9. MATLAB *Reference Guide*, Math Works, (2006).
10. Microchip web site: www.microchip.com
11. Analog Devices Inc. web site: www.analog.com
12. C. Marndven and G. Ewers, *A simple approach to digital signal processing*, Wiley, (1996).
13. N. Ahmed and T. Natarajan, *Discrete-time signals and systems*, Prentice-Hall, Englewood Cliffs, (1983).
14. L. B. Jackson, *Digital filters and signal processing*, Kluwer Academic publishers, (1989).
15. S. K. Mitra, *Digital signal processing: a computer-based approach*, McGraw-Hill, (1998).
16. A. E. Sarsour, *Microcontroller based digital filter development*, MSc Thesis Near East University, Dept. Computer Eng., (2006).
17. V. K. Ingle and J.G. Proakis, *Digital signal processing using MATLAB*, Boston PWS publishing, (1997).
18. Near East University web site: www.neu.edu.tr
19. Hi-Tech web site: www.htsoft.com
20. Cleverscope *User Guide*, www.cleverscope.com

Dogan Ibrahim, a Professor of Computer Engineering at the Near East University in Cyprus, has been actively involved in microprocessor and microcontroller based systems for the last 17 years. His research interests include the design of microcontroller based automation systems, digital signal processing, and distant engineering teaching. He teaches courses in digital logic, microcontroller systems, and automatic control theory.