BASIC TECHNIQUES LET DESIGNERS BUILD A FINITE-IMPULSE-RESPONSE FILTER IN DEDICATED HARWARE USING PROGRAMMABLE LOGIC.

LEARN THE FUNDAMENTALS OF DIGITAL FILTER DESIGN

Historically, designers often have taken an analog approach to filtering. Filters were constructed using operational amplifiers, resistors, and capacitors. One op amp could implement a second-order filter, and higher-order filters could be implemented by cascading second-order filters. However, passive components with tolerances of 1% or better are necessary for the filter to have reproducible characteristics. And the filter is typically fine-tuned by trial-and-error substitution of available component values. In addition, operational amplifiers with a high gain-bandwidth product may be needed to keep undesirable phase shift to a minimum or keep a closed-loop system stable. These factors are among the many problems in real-world implementations of filters.

With the advances made in digital-signal processing, however, digital filters are becoming a more attractive design alternative to traditional analog techniques. Because digital-system information is in digital form, filtering can be accomplished relatively easily by passing the data through a filter algorithm. In addition, digital filters have the advantages of no filter-characteristic drift over time, temperature, or voltage. And they can easily be designed to filter low-frequency signals. Moreover, the filter response can be made to closely approximate the ideal response, and linear phase characteristics are possible.

There are many well established methods of determining the filtering algorithm. Basically, the designer establishes the desired filter characteristics, thereby yielding a filter transfer function. The continuous-time transfer function is then transformed to the equivalent linear discrete-time-difference function. This function in the Z domain has the general form of:

$$G(Z)= (A_0 + A_1Z^{-1} + A_2Z^{-2} + \dots A_nZ^{-n}) / (1 + B_1Z^{-1} + B_2Z^{-2} + \dots B_mZ^{-m}) = Y(Z)/X(Z)$$

The equation is referred to as the pulse transfer function. It's actually the Z transform of the continuous-time filter's unit impulse response. Conversely, the inverse Z transform of the pulse transfer function yields the impulse response of the filter.

The coefficients A_n and B_m determine the response of the digital filter. Changing the coefficients changes the response of the filter. The terms Z^{-n} and Z^{-m} represent sampling delays or taps. The G(Z) equation represents the algorithm of sampling the input, multiplying it by A_0, and adding it to the previous sample that's been multiplied by A_1, then adding that value to the next previous sample which has been multiplied by A_2, and so on. An output value occurs when all N values have been multiplied and accumulated.

In parallel, each output value is stored, multiplied by B_1, then added to the previous output value which has been multiplied B_2, and so on. The equation can be rearranged so that the result of the output multiply accumulate is added to the result of the input multiply accumulate to produce an output. This procedure is referred to as convolution. An output sample is produced for every input sample *(Fig. 1)*.

The key to digital-filter design is to determine the filter coefficients that will produce the desired frequency response. Recursive digital filters, or infinite-impulse-responsive (IIR) filters, are a type of digital filter in which the design methodology closely follows that of an analog filter. One method for determining the coefficients is to define a realizable continuous-time domain Chebyshev, Butterworth, or equal-ripple filter then use Z transforms to transform the continuous-time-domain transfer function to the equivalent discrete-time transfer function that yields the filter coefficients.

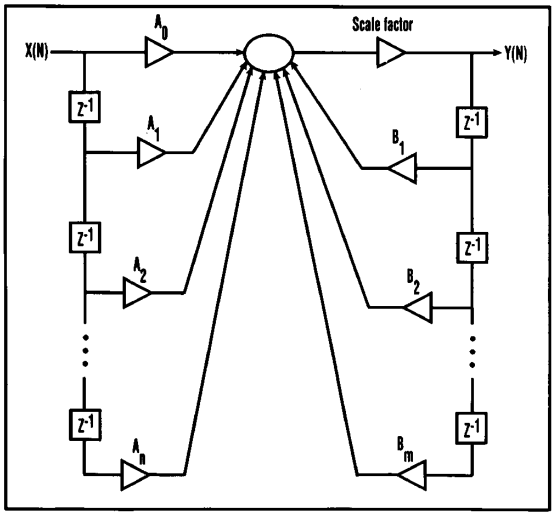
A second popular method is the bilinear transform. In this method, engineers first design an analog filter so that after it's transformed to a digital filter, the resulting filter meets a set of desired digital-filter specifications. This analog filter is then transformed to a digital filter via the bilinear transform from the S variable of the Laplace transform to the Z variable of the Z transform.

In a non-recursive digital filter or finite-impulse-response (FIR) filter, the output is computed using the present input X_n and the previous inputs $X_{n-1}, X_{n-2} \dots X_{n-N}$. This implies that the coefficients, B_m, are all 0, and there's no feedback from the output. Designing non-recursive digital filters (FIR) involves defining an ideal desired frequency response from which the ideal impulse response is computed. The ideal impulse response is truncated to a finite number of non-zero samples using a windowing function, which is judiciously chosen. A common windowing function is the Kaiser window function. An interesting property of FIR filters is that if an FIR system has linear phase, then its frequency response is constrained to be zero at $f = 1/2T$, where T equals the sampling frequency if:

$h[M - n]=h[n]$ and M is odd. (M = truncation length of the window).

This implies the M should be even when designing high-pass and bandstop filters. Or,

$h[M - n]= -h[n]$ and M is even.



1. IN THE FUNCTIONAL structure of a digital filter, the A and B coefficients determine the response of the filter and the Z terms represent sampling delays called taps.

A second method is the Parks-McClellean method. In this approach, the filter order and the edges of the passbands and stopbands are fixed, and the impulse-response coefficients are varied systematically so that an equal-ripple behavior is achieved in each approximation band. With this approach, the filter order can't be specified in advance. Therefore, a cut and try procedure must be used to find the minimum filter order. The cut and try can be reduced by using a formula that predicts the filter order required to meet a given set of specifications.

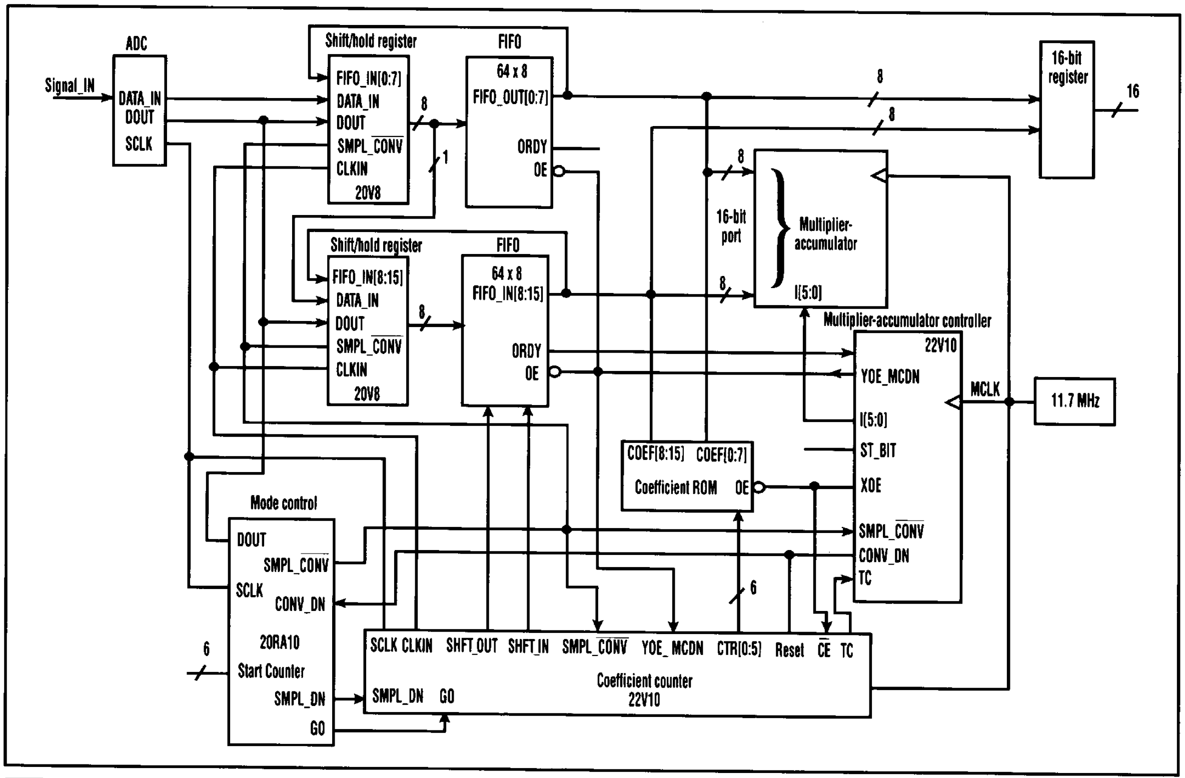
There are advantages and disadvantages to each type of digital filter (IIR and FIR). An FIR filter is always stable because there's no feedback from the output and the impulse response is finite. In addition, the amplitude and phase can be arbitrarily specified. On the other hand, an FIR filter will generally require more taps, and consequently more math, to compute the output value. The design methodology doesn't resemble the familiar analog design techniques.

An IIR will generally have fewer coefficients, but the required output feedback can make circuit implementation more complex. A stable IIR filter can become unstable if the coefficients aren't chosen properly to account for digital math errors.

There are four main type of errors that can arise in the design of digital filters. These are referred to as quantization errors. They are:

1. Quantization errors of the input analog-to-digital conversion
2. Quantization errors of the coefficients
3. Quantization errors due to arithmetic computations, including overflow
4. Limit cycles

In most cases, a 12-bit analog-to-digital converter (ADC) provides enough dynamic range and sufficiently small quantization noise. If floating-point numbers are used for the filter coefficients, the quantization error is usually small enough. However, floating-point arithmetic is more complex and more expensive



2. AN FIR FILTER IS IMPLEMENTED in a circuit that uses a single-port 16-bit multiplier-accumulator capable of a 85-ns clock speed. Because it's based on microcode, the multiplier-accumulator can be controlled with a PLD.

to implement than integer or fixed-point arithmetic. If 12- or 16-bit coefficient are used, the quantization error is generally negligible.

In the digital domain, math is performed using finite precision binary arithmetic. All digital filters need to multiply a signal sample by a constant coefficient. Of course, multiplying 2 N-bit binary numbers results in a 2N-bit result, but digital systems are usually confined to a fixed number of bits with which to represent binary numbers. Therefore, it's necessary to round off the 2N-bit digital number back to N bits. If a 32-bit multiply accumulator is used and the final output is rounded to 16 bits, the arithmetic quantization errors can be minimized.

If overflow occurs during mathematical operations, the digital filter can behave in a nonlinear fashion and oscillations can occur. Twos-compliment arithmetic can help eliminate overflow. In addition, a saturating adder can be used. If the coefficients are less than one, then the resulting product will also be less than one. Scaling is used to force this condition. The coefficient can be scaled by a multiple of two so that the largest coefficient uses all available bits in the binary representation. The input is then scaled by the same amount.

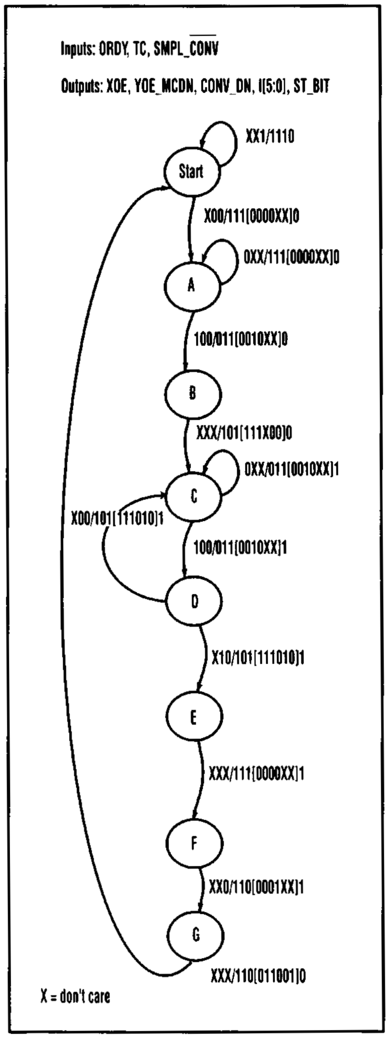
The detail with which a digital filter can be described can seem endless. Fortunately, a wide variety of computer programs exist that help the engineer with the filter's design. One such product is the DFDP software from Atlanta Signal Processing Inc. (ASPI), Atlanta, Ga.

Before a signal can be digitally filtered it must be digitized by an ADC. If a delta-sigma converter is used, the need for antialiasing filters (which must be analog and can be many orders) is virtually eliminated. Delta-sigma converters may have sample rates as high as 100 kHz. The filter algorithm can then be implemented in software or hardware.

A single-chip microprocessor can be used to implement a digital filter in software. However, "single chip" may be misleading, because a microprocessor system will generally require system RAM, ROM, I/O, and glue logic. The microprocessor can implement low- to medium-performance digital filters if the only function they're performing is the digital filtering. As the work load of the microprocessor increases, its capability to digitally filter a signal in real time decreases. Once the system is designed, changing the filter's characteristics is as easy as changing variables in software and downloading the code to the system.

For higher performance and moderate flexibility, the filter can be implemented in dedicated hardware using programmable logic for design flexibility. The limiting parameter will be the time to do a multiply-accumulate function and the amount of physical space required for the hardware implementation of the taps. Consider a circuit that uses a single-port 16-bit multiplier-accumulator capable of an 85-ns clock speed *(Fig. 2)*. The device can work in twos-compliment numbers and has output saturation capabilities. As stated before, these two features are desirable when implementing digital filters. In addition, the device can be easily controlled with a programmable logic device (PLD) because it's microcoded based.

First, the system must initially load the first N (N = 64) samples into the FIFO before any convolution takes place. Otherwise, the FIFO would never fill up. A counter implemented in a 20RA10 works well. The 6-bit counter is implemented with the four least-significant bits implemented as an asynchronous counter. SMPL_DN (ADC sample done) acts as the clock. The two most-significant bits are implemented as a ripple counter. This type of counter design makes it possible for a long counter to be implemented with only four product terms per output. The SMPL_DN signal is also generated in the 20RA10, and is triggered off signals from the ADC.



3. AN 8-STATE state machine implements the operations of loading a sample into the multiplier-accumulator, then loading the coefficients in and issuing the multiply-accumulate command until all N samples are done.

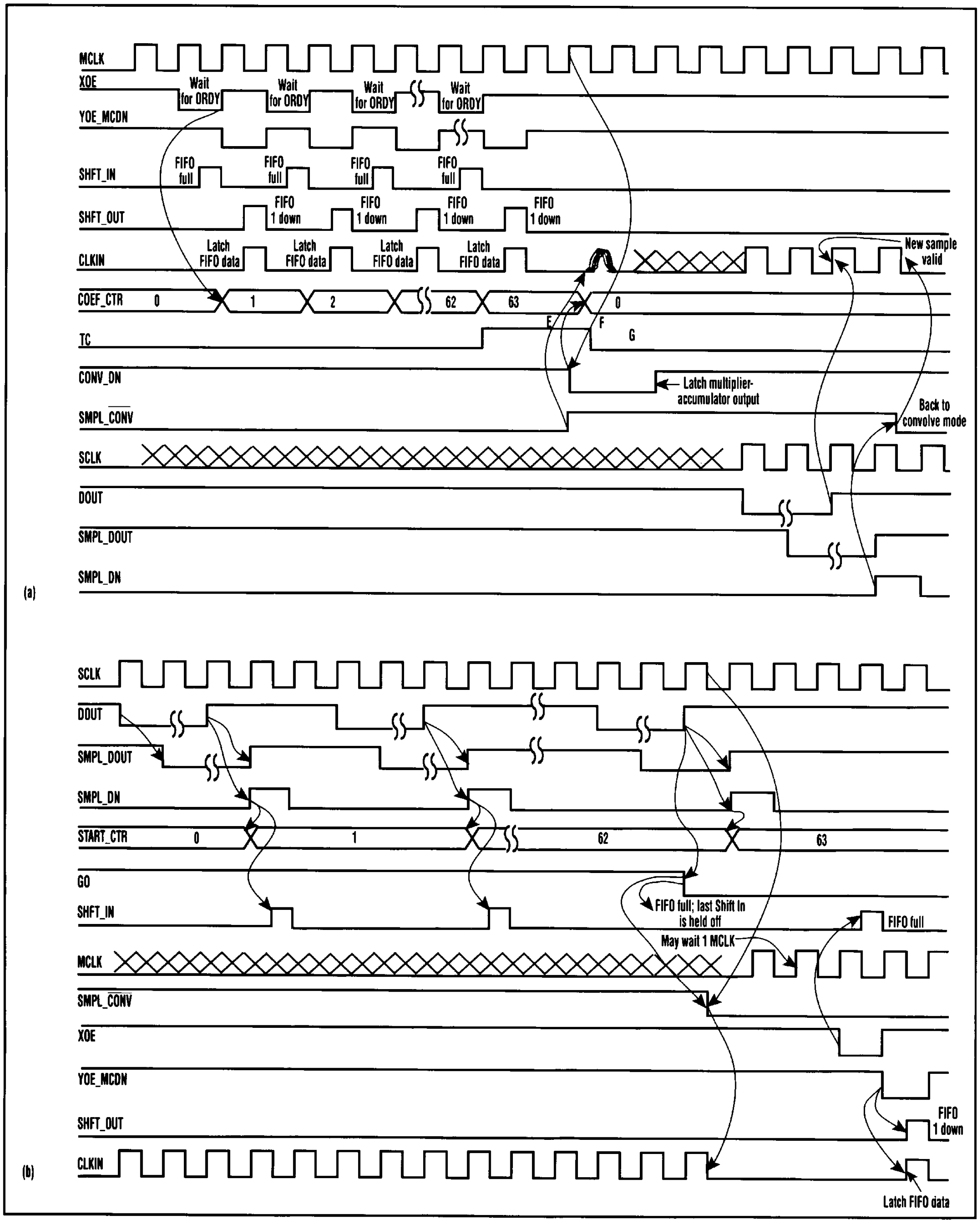
When the counter reaches the value 63, indicating that the FIFO is full minus the one sample that's held in the shift/hold register, GO becomes true and the system begins to execute the filtering algorithm. Because the system is linking two asynchronous subsystems (ADC and the multiplier-accumulator), there must be an asynchronous interface between the two. The 20RA10 is utilized by generating one interface signal SMPL_$\overline{\text{CONV}}$ (sample or convolve mode). The system powers up with this line held in the sample mode (SMPL_$\overline{\text{CONV}}$ = 1). When GO goes true, synchronous with the falling edge of the clock from the ADC, SMPL_$\overline{\text{CONV}}$ goes low asynchronously with MCLK (synchronous with SCLK). Because SMPL_$\overline{\text{CONV}}$ is an input to the state machine, the machine could be subject to a metastable input. The Lattice CMOS PLDs are very high speed, so the metastable characteristics are excellent. That is, the state flip-flop has a very low probability of going metastable. Therefore, the state machine will have to wait, at most, one extra MCLK cycle before starting the convolution.

Once the convolution is started, the operations of loading a sample into the multiplier-accumulator, then loading the coefficient into the multiplier-accumulator and issuing the multiply-accumulate command, can be repeated until all N samples have be done. At this time, the filter output is valid and the cycle is restarted. These steps can be implemented with an 8-state state machine (multiplier-accumulator controller) *(Fig. 3)*.

By coding the states properly, the state variables out of the state machine can be used to directly control the multiplier-accumulator. Two output enable signals, XOE and YOE_MCDN, control the data into the multiplier-accumulator. The signal CONV_DN indicates that all N samples have been convolved. A dummy state variable (ST_BIT) is used so that the state bit (XOE, YOE_MCDN, CONV_DN) can be employed as outputs. If the dummy bit was unused, two states would be forced to have the same state assignments, which isn't allowed. The design takes advantage of the power-up reset of Lattice's programmable logic devices (PLD s). After power-up, the registers will be left in the 0 state, which by careful design is also the start state of the state machine.

Except for the last SMPL_DN during initial load, every time SMPL_DN (sample done by the ADC) takes place, SHFT_IN occurs to load sampled data from the shift/hold registers into the FIFO. During convolution, XOE occurs every time a coefficient is loaded to the multiplier-accumulator. The first XOE of a convolution causes the last data sample left in the shift/hold registers during initial load or sample mode to be shifted into the FIFO. Following every XOE is a YOE_MCDN (Y-output enable, multiply-accumulate done). YOE_MCDN causes data from the FIFO's output to be parallel loaded into the shift/hold registers. A single data sample is then shifted out of the FIFO. The system is ready for the next XOE that shifts in the data held in the shift/hold registers and so on. This loop continues until SMPL_$\overline{\text{CONV}}$ (sample or convolve mode) goes to sample mode, at which time a new sample is loaded into the shift register, restarting the cycle.

Inputs to the state machine,



4. FIFO CONTROL SIGNALS are generated asynchronously. The system timing diagrams for the convolve (a) and initial load (b) operations show the appropriate Shift In and Shift Out signals, and clock signals sent to the shift/hold register.

SMPL_$\overline{\text{CONV}}$, tell the machine when it's time to begin the convolution cycle. This signal comes from the mode-control device. TC (Terminal Count) indicates when the convolution is to end. TC comes from a 6-bit coefficient counter, and is valid when the count equals 63, which indicates when all 64 samples have been convolved with the respective coefficients. ORDY comes from the FIFO and tells the state machine that the sample from the FIFO is valid. The state machine will continue to load in the coefficient to the multiplier-accumulator until ORDY goes true, at which time the state machine will advance to the next state. If the cycle time of the multiplier-accumulator never exceeds the access time of the FIFO, ORDY should always be true when it's an input the state machine depends on.

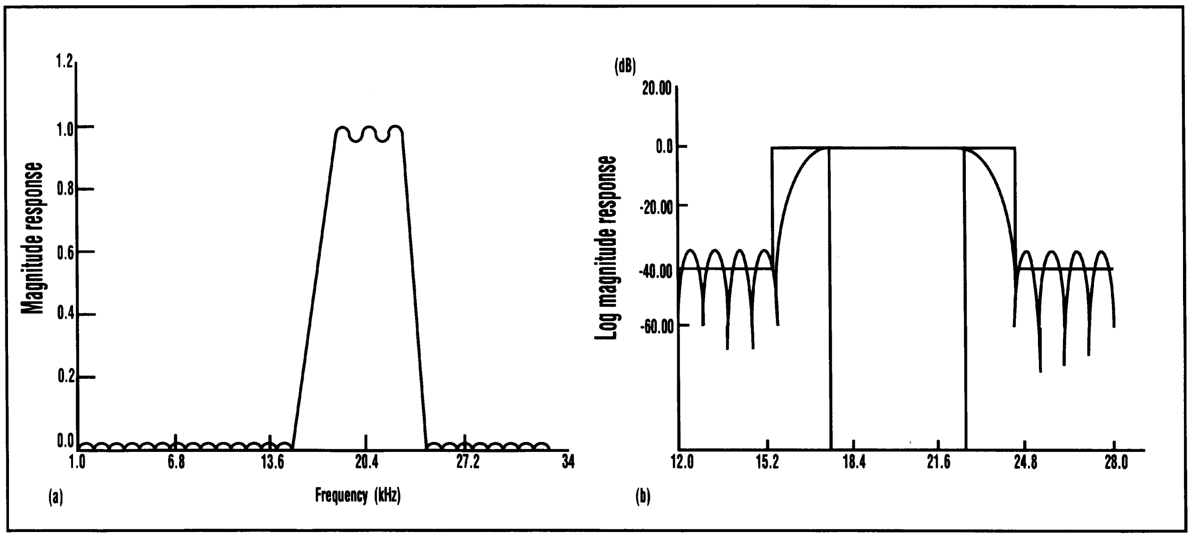
Microcoded instructions to the multiplier-accumulator are generated by decoding the state variables. The first instruction is a NOOP. When SMPL_$\overline{\text{CONV}}$ goes low, then state machine issues a XBUS instruction to the multiplier-accumulator. This causes the multiplier-accumulator to load data from the I/O port into an internal register. The state machine then issues a YBUS; CLKMR TC. This command tells the multiplier-accumulator to perform a multiply operation in twos-compliment without accumulation because it's the first multiply operation of the convolution.

The machine then enters a loop and issues another XBUS command followed by a YBUS; CLMR; TC; MR+. This command is a multiply-accumulate function in twos-compliment arithmetic. The machine remains in this loop until TC goes true, at which time the last multiplier-accumulator cycle is completed and the output command MS (SAT) is issued. MS causes the filter's outputs (multiplier-accumulator outputs) to become valid and latched into a final output register. This command will saturate the multiplier-accumulator output if the final value has an overflow, keeping the digital filter from oscillating. The multiplier-accumulator is statically configured to round off the final output to the most significant 16 bits.

The instructions to the multiplier-accumulator can be changed simply by decoding the state variables to different output values. If E^{2}CMOS devices are used, the programmable device can simply be reprogrammed and put back into the circuit. An E^{2}CMOS 22V10 from Lattice Semiconductor is one such device that can be used for this application.

Two 64-word-by-8-bit FIFOs can be used to implement the filter taps. The FIFO can be loaded up with the initial N samples. A sample is then shifted out of the FIFO and into the multiplier-accumulator for processing. This sample is also stored in a shift/hold register and is shifted back into the FIFO prior to the next sample being shifted into the multiplier-accumulator for processing. After all N samples have been processed, the oldest sample is shifted out and a new ADC sample shifted in. The multiplier-accumulator can then output a filter value. Programmable logic can be used to interface the digital filter to the ADC, act as temporary storage register, and implement FIFO control.

These shift/hold registers can be implemented with two 20V8 devices. In the sample mode (SMPL_$\overline{\text{CONV}}$ = 1), the devices act as shift registers. Data is serially loaded into them under control of the ADC. The registers are then placed in a hold mode so that the data sample isn't lost. When the system enters the convolve mode, (SMPL_CON = 0), data is immediately loaded into the



5. A PLOT OF THE MAGNITUDE response shows that the bandpass filter's center frequency is 20 kHz with a passband of 5 kHz (a). The transition region occurred in 2 kHz. The log magnitude response plot reveals a 175-dB/decade slope at the edges of the filter (b). It would take a 9th-order analog filter to implement the same specifications.

shift/hold registers in parallel.

Filter coefficients are stored in PLDs emulating ROM. A 6001 has a programmable AND and a programmable OR array so that it easily emulates a 64-by-8 high-speed PROM. Again, if E^2 devices are used, the filter coefficients can be changed simply by reprogramming the devices. An address counter is used to access the coefficients in the correct order. Because there are 64 required coefficients for the 64 taps, only 6 bits of address are required.

The coefficient-address counter is a simple 6-bit counter implemented in a 22V10. The counter is a synchronous type with a count enable. The clock is synchronous with the multiplier-accumulator clock. The count-enable input pin is connected to XOE from the multiplier-accumulator controller. Therefore, the counter is incremented only after the coefficient value has been loaded into the multiplier-accumulator. When the counter reaches 63, TC goes true to indicate that all 64 coefficients have been convolved. Again, the power-up reset is used to ensure that the counter starts in a known state.

The remaining four output-logic macro cells can be used to generate FIFO control signals. These signals are generated asynchronously. Depending on the state of the system—whether it be initially loading, sampling, or convolving—the appropriate Shift In, Shift Out, and clock signals for the shift/hold register will be generated *(Fig. 4)*.

When the convolution is done, the state machine sets the CONV_DN signal true synchronous with MCLK. Hence, SMPL_$\overline{\text{CONV}}$ will also be set synchronous with MCLK. This will create glitches on the signal CLKIN, which is the clock to the shift/hold registers. This is a don't-care condition, as the registers will soon be loaded with a new valid data sample under the control of the ADC.

The system requires 133 MCLK cycles to complete the convolution. With a 11.7-MHz clock, this takes 11.4 μs. This system used an ADC with a serial interface that requires 3.3 μs to shift the data into the shift/hold registers. Thus, the system can sample an input signal at 11.4 + 3.3 = 14.7 μs or 68 kHz. The Nyquist sampling theorem states that a signal must be sampled at twice the highest frequency component to accurately preserve the information in that signal. Therefore, this system can accurately filter a signal with the frequency component as high as 34 kHz.

Using the DFDP software from ASPI, a bandpass filter was designed using the Parks-McClellean method. The center frequency is at 20 kHz with a passband of 5 kHz. The transition region occurred in 2 kHz *(Fig. 5)*. It's interesting to note that the edges of the filter have a slope of approximately 35 dB/0.2 decade, or 175 dB/decade. It would take a 9th-order analog filter to implement the same specifications.

The system presented in this example is a straightforward FIR filter. Because of the extensive use of programmable logic, the system can be easily adapted to implement an IIR filter. The final output value can be fed back into the FIFO prior to a new sample shifting into the FIFO. The coefficients can be staggered in the coefficient ROM so that the B_ms line up with the Y(n - M), and the A_ns line up with the X(n - N).

If enhancement of the system's performance is desired, a larger FIFO memory can be used with a faster multiplier-accumulator. Because 15-ns programmable-logic devices are used, they're not a limiting factor. If a parallel ADC, 64-by-8 FIFO, and a 45-ns multiplier-accumulator are employed, the system could be made to run at 167 kHz with little modification.☐

The author would like to thank Atlanta Signal Processing for their help in developing this article.

Lattice Semiconductor Corporation (LSC) products are made under one or more of the following U.S. and international patents: 4,761,768 US, 4,766,569 US, 4,833,646 US, 4,852,044 US, 4,855,954 US, 4,879,688 US, 4,887,239 US, 4,896,296 US, 5,130,574 US, 5,138,198 US, 5,162,679 US, 5,191,243 US, 5,204,556 US, 5,231,315 US, 5,231,316 US, 5,237,218 US, 5,245,226 US, 5,251,169 US, 5,272,666 US, 5,281,906 US, 5,295,095 US, 5,329,179 US, 5,331,590 US, 5,336,951 US, 5,353,246 US, 5,357,156 US, 5,359,573 US, 5,394,033 US, 5,394,037 US, 5,404,055 US, 5,418,390 US, 5,493,205 US, 0194091 EP, 0196771B1 EP, 0267271 EP, 0196771 UK, 0194091 GB, 0196771 WG, P3686070.0-08 WG. LSC does not represent that products described herein are free from patent infringement or from any third-party right.

The specifications and information herein are subject to change without notice. Lattice Semiconductor Corporation (LSC) reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. LSC recommends its customers obtain the latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

LSC warrants performance of its products to current and applicable specifications in accordance with LSC’s standard warranty. Testing and other quality control procedures are performed to the extent LSC deems necessary. Specific testing of all parameters of each product is not necessarily performed, unless mandated by government requirements.

LSC assumes no liability for applications assistance, customer’s product design, software performance, or infringements of patents or services arising from the use of the products and services described herein.

LSC products are not authorized for use in life-support applications, devices or systems. Inclusion of LSC products in such applications is prohibited.

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996