Ray Mack, W5IFS

17060 Conway Springs Ct, Austin, TX 78717; **w5ifs@arrl.net**

SDR: Simplified

Filter Design Program

Wes Hayward,W7ZOI, Rick Campbell, KK7B, and Bob Larkin, W7PUA, present a short description of DSP techniques and filtering in *Experimental Methods in RF Design*.[1] One of the best items included with the book is a BASIC program (written by Bob Larkin, W7PUA) on the CD, that will calculate the coefficients for a finite impulse response (FIR) filter. It is a very well written program, but not terribly useful in a modern world without *QBasic* or *GWBasic* programs on our computers. Luckily, I still have a version of BASIC that came with *MS-DOS 3.0* for the original IBM PC back in the 80s that I used to verify my port of the program to *C*.

One of the programs in the 5x12_Mack_SDR.zip file for this installment is the program that I have written that implements the BASIC code in *C*.[2] The present incarnation is designed to create a *C* source file with the FIR filter coefficients and related information. The logic of the program is entirely Bob's, but even Bob used code from another source. In making sure I had the new program correct, I verified the Bessel calculation with the source Bob used from the second edition of *Numerical Recipes in C*.[3] This is a fantastic book that you can read in its entirety on the web. To paraphrase Newton, "we see farther because we stand on the shoulders of giants."

The Gibbs Phenomenon

The program starts with a rectangular shaped ideal filter. Figure 1 shows such a low pass filter with a 1000 Hz cutoff and a 8000 Hz sample rate. The coefficients of an FIR filter are generated by calculating the Fourier transform of the filter frequency response to determine the impulse response in the time domain. For all but the simplest filter shapes, the Fourier transform can get pretty messy. It turns out that the Fourier transform and impulse response for the low pass, high pass, and band pass response is the same rather simple equation with different parameters based on the cutoff frequencies. In essence, all three filter types are variations of a rectangular pulse shape.

Let's choose to create a filter with

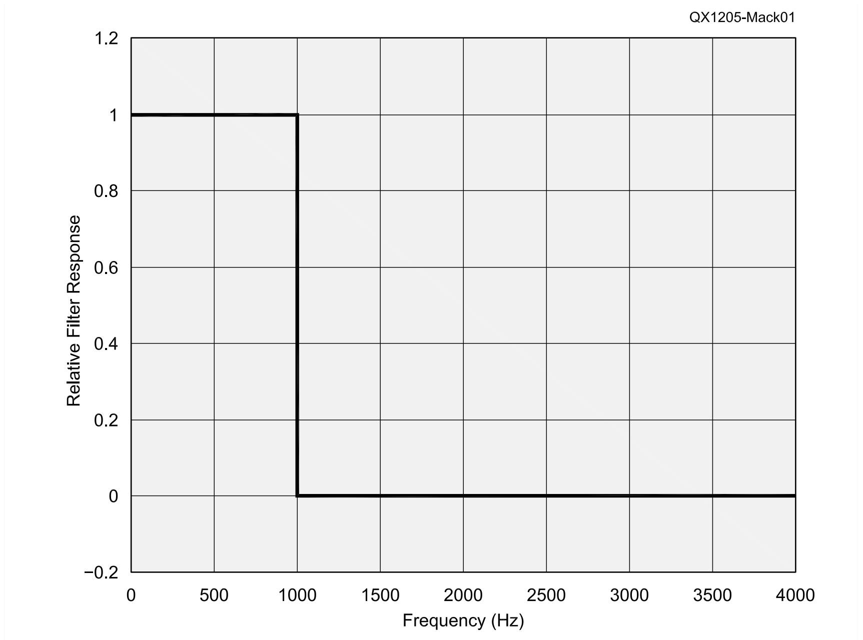


Figure 1 — This graph shows the ideal filter response of a 1000 Hz low pass filter with 8000 Hz sample rate.

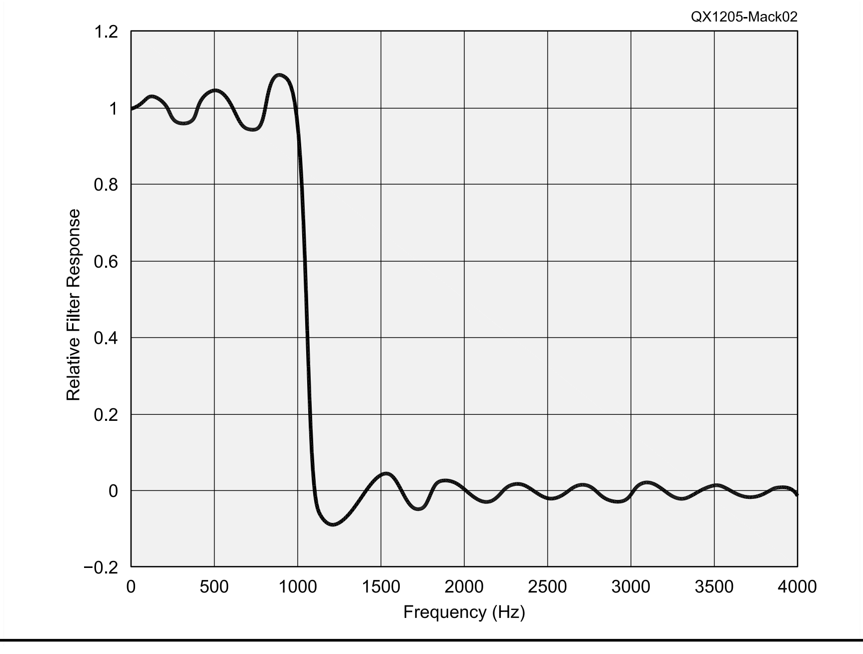


Figure 2 — Here is the actual frequency response of an "ideal" FIR filter with 1000 Hz cutoff, 8000 Hz sample rate and 20 coefficients. The filter response is shown with linear scaling rather than the magnitude in dB.

[1]Notes appear on page 44.

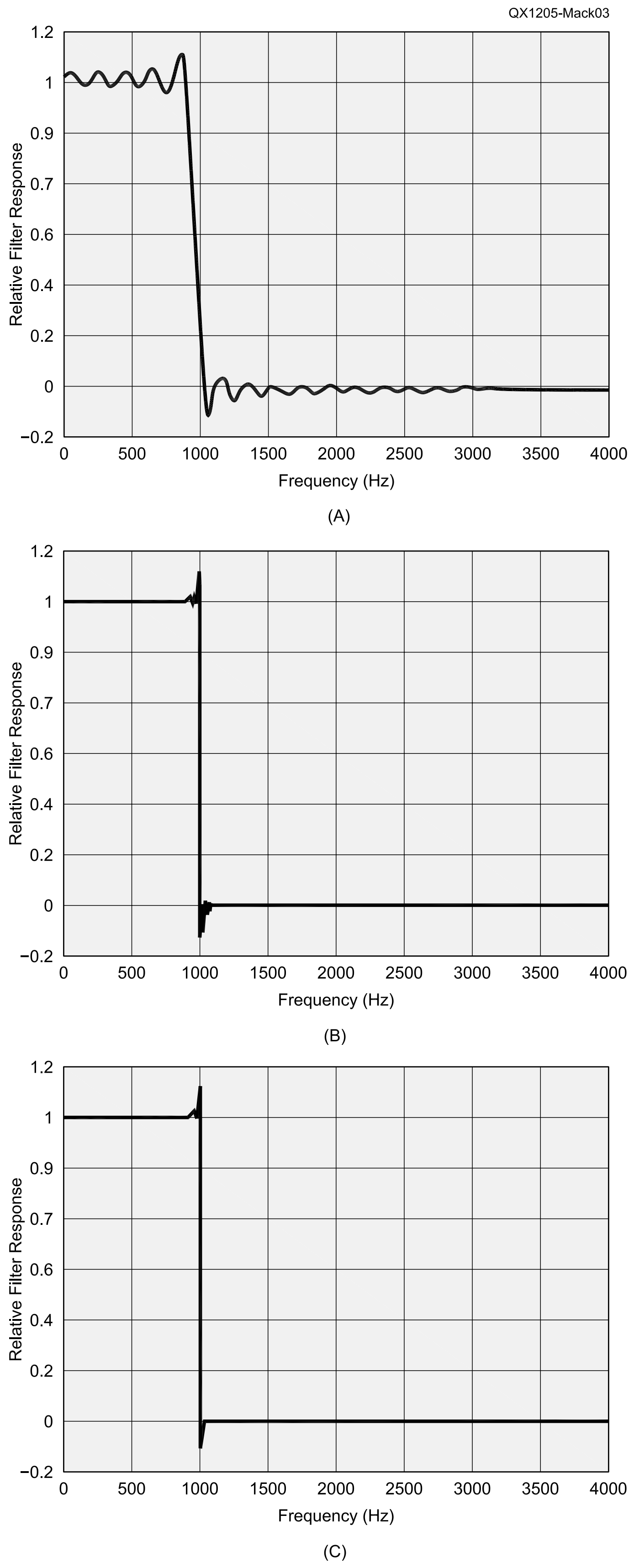


Figure 3 — Part A shows an "ideal" low pass filter with 40 coefficients. Part B shows the same filter, but with 200 coefficients. Part C shows the same filter, but with 1000 coefficients. You can see that each filter has the characteristic 8.9% overshoot and undershoot.

From **MILLIWATTS** *to* **KILOWATTS**℠

More Watts per Dollar℠

Taylor TUBES

Quality Transmitting & Audio Tubes

Eimac®

- **COMMUNICATIONS**
- **BROADCAST**
- **INDUSTRY**
- **AMATEUR**

Svetlana

Immediate Shipment from Stock

3CPX800A7	3CX15000A7	4CX5000A	813
3CPX5000A7	3CX20000A7	4CX7500A	833A
3CW20000A7	4CX250B	4CX10000A	833C
3CX100A5	4CX250BC	4CX10000D	845
3CX400A7	4CX250BT	4CX15000A	866-SS
3CX400U7	4CX250FG	4X150A	872A-SS
3CX800A7	4CX250R	YC-130	5867A
3CX1200A7	4CX350A	YU-106	5868
3CX1200D7	4CX350F	YU-108	6146B
3CX1200Z7	4CX400A	YU-148	7092
3CX1500A7	4CX800A	YU-157	3-500ZG
3CX2500A3	4CX1000A	572B	4-400A
3CX2500F3	4CX1500A	807	M328/TH328
3CX3000A7	4CX1500B	810	M338/TH338
3CX6000A7	4CX3000A	811A	M347/TH347
3CX10000A7	4CX3500A	812A	M382

– TOO MANY TO LIST ALL –

VISA MasterCard DISCOVER

ORDERS ONLY:

800-RF-PARTS • 800-737-2787

Se Habla Español • We Export

TECH HELP / ORDER / INFO: 760-744-0700

FAX: 760-744-1943 or 888-744-1943

An Address to Remember:

www.rfparts.com

E-mail:

rfp@rfparts.com

RF *Since 1967*

RF PARTS™ COMPANY

20 coefficients. The resulting filter response looks something like Figure 2. You see that the pass band has ripples every 200 Hz and the stop band has ripples also spaced every 200 Hz. The ripples don't look very big when plotted on a linear scale. They are pretty serious when you plot them as dB, however. These ripples are called the Gibbs Phenomenon (first discovered in 1848 and named for J. Willard Gibbs who described the phenomenon in detail in 1899). The short description of the phenomenon is that any discontinuity in one domain causes an infinite series in the other domain. In the case of our filter, the sharp discontinuity at 1000 Hz in the frequency domain requires an infinite number of coefficients in the time domain to implement that frequency spectrum. Since we cannot implement our filter in the time domain with an infinite number of coefficients, the 20 coefficients create 20 discrete bins each 200 Hz wide (4000 Hz / 20). If we had chosen a 40 coefficient filter, the size of the ripples in the pass band and stop band would be smaller in both width (now only 100 Hz wide) and height because the sum of the series is closer to convergence. The longer filter is a closer approximation to the original function. Figure 3 shows how increasing numbers of coefficients increase the slope of the transition but that even very large numbers of coefficients will not eliminate the issues right at the edge of the transition region. Gibbs found that an FIR filter will have 8.9 % maximum ripple for the first ripple on either side of the transition, regardless of the number of filter coefficients.

Table 1 — Frequency Register Values for the Allowed Sample Rates

Rate	*R*	*J*	*D*	*P*	*MADC & NDAC*	*DOSR & AOSR*	*PLL Frequency*
8000	1	1	792	2	2	128	10.752
16000	1	1	792	1	2	128	21.504
24000	1	3	584	1	2	128	43.008
48000	1	7	1680	1	2	128	86.016
96000	1	7	1680	1	1	128	86.016
192000	1	7	1680	1	1	64	86.016

Listing 1 — The code for the main signal processing loop.

```
while (1)
{
   /* Read 16-bit left channel Data */
   EZDSP5535_I2S_readLeft(&data1);
   /* Read 16-bit right channel Data */
   EZDSP5535_I2S_readRight(&data2);
   // perform the IF filtering
   error = fir(&data1, coefficients, &data1,
               delay_buffer, 1,
               number_of_coefficients);
   //Do the demodulation
   switch (modulation_type)
   {
      case AM:   // square law detector
         demod_sample *= demod_sample;
         break;
      case FM:
         demod_sample = demod_FM(data1);
         break;
      case CW:
         demod_sample = demod_CW(data1);
         break;
      case LSB:
         demod_sample = demod_LSB(data1);
         break;
      case USB:
         demod_sample = demod_USB(data1);
         break;
   }
   // perform the baseband filtering
   error = fir(&demod_sample, baseband_coefficients,
        &demod_sample, demod_delay_buffer,
         1, number_of_baseband_coefficients);
   /* Write 16-bit left channel Data */
   EZDSP5535_I2S_writeLeft(data1);
   /* Write 16-bit right channel Data */
   EZDSP5535_I2S_writeRight(data2);
   // If any key has been hit in the debugger, we exit
   if (_kbhit())
   {
      break;
   }
}
```

The Kaiser Window

Gibbs observed that as a function becomes smoother, the coefficients of the transform near the center become much larger and coefficients further away quickly tend toward zero. There are two ways to force the frequency response to have fewer ripples and approach a smooth shape. The first is to design a filter that is not ideal, has sloped shoulders, and a gradual transition from pass band to stop band. The problem with this approach is that it requires using the Fourier Transform to calculate the coefficients and the results are unique for each filter. The second way is to start with an ideal response, with its simple calculations, and then force the coefficients to have a shape that has significant central coefficients but with the coefficients near the edges rapidly approaching zero. This process is called windowing. There are many functions that can be multiplied against the ideal filter coefficients to achieve varying amounts of pass band or stop band ripple reduction or both. The transition rate from pass band to stop band is also affected. Each window method has its own set of advantages and disadvantages. As with most other situations in engineering, you can affect stop band or pass band or transition rate: pick two!

The *C* program uses a Kaiser window that applies a Bessel function to the coefficients to set a desired amount of stop band attenuation close to the pass band in exchange for a more gradual transition band and a flat pass band. The Kaiser window is named after J. F. Kaiser, who decided to use some very obscure (even for mathematicians) and difficult to calculate functions called prolate spheroidal functions. Kaiser windows are probably some of the best for controlling the depth of the first side lobe while still giving a rapid transition. Fortunately, Bob Larkin handled the nasty math in his original program.

More TI Software Resources

TI gives away a package called DSPLIB that can be used for any of their DSP families. Go to **www.ti.com/lit/ug/spru422j/spru422j.pdf** to download the Programmer's Reference. Then go to **www.ti.com/tool/sprc100** to download the zip file containing the library and its source code. You need to place the header file dsplib.h in the **ccsv4\tools\compiler\include** directory. Place all of the library files in the **ccsv4\tools\compiler\lib** directory. You will need the source files because the library is compiled in the small memory model and the other libraries are in large memory model. For this program, you need **fir.asm** in your project directory.

We are interested in the function "fir." You will find the reference information on page 4-46 of the reference manual. The function takes six arguments. The first argument is the address of the array of input samples. The second is the address of the array of filter coefficients. The next one is the address of the output buffer, which can also be the address of the input buffer for computation in-place. The fourth argument is the address of the delay buffer, which holds all of the history of the filter. This buffer is equal in size to the number of coefficients plus one more that holds the array index of the oldest entry. The C language does not include the size of an array as part of the array storage. That means we have to keep track of the size as another piece of data. The "fir" function uses the fifth argument to hold the size of the input array and the sixth is the size of the filter coefficient array. All of the arguments are 16 bit signed or unsigned numbers.

Figure 4 shows the concept of an FIR filter. In fact, this structure applies to any FIR operation where "filter" can encompass any manipulation such as a Hilbert transform (90° phase shift with no amplitude change). The figure shows a trivial example where the delay line (a shift register when implemented in hardware) starts with all registers holding zero. It goes through the first six sample periods showing the data in the delay line and the calculations that occur. The data samples are (–1, –2, –1, 2, 5, 10) and the six coefficients are (–1, 2, 6, 6, 2, –1).

The code for this experiment uses the fir() function in its single element mode. The data word is applied to all of the delay elements and the total is added together to produce a single output word. The intermediate history is held in the delay_buffer array that is declared inside our automatically created data file. The documentation calls for the array to be initialized to all zeros, but that occurs automatically as part of the C startup when delay_buffer is copied into memory.

The Bare Metal Filter and Receiver Program

The last step in making a radio with a band pass filter is to port the filter coefficient calculation into our receiver so that we can tune the filter to any signal within the pass band. The biggest risk in moving the FIR filter coefficient task into our DSP is running out of program memory. The FIR coefficient calculation uses floating point and numerous math library functions that consume quite a lot of program memory.

In addition to calculating the coefficients, we need to set the PLL to achieve the design sample rate. We saw in the Mar/Apr issue that we need to calculate both an integer and fractional value to set the sample rate. Here is the equation:

$$PLL\_FREQ = \frac{PLLCLK \times (R \times J.D)}{P}$$

The PLLCLK value is 12 MHz [Remember from the last issue that the dot between the J and the D is the notation for the multiplication factor. If R = 1, P = 1, J = 7 and D = 1680, then the expression (R × J.D) = (1 × 7.1680). — Ed.] Fortunately, the data sheet gives the values for 12 MHz MCLK to generate 48 kHz and 44.1 kHz so we only have to do the calculations for the other allowed sample rates. Table 1 captures the register values for the sample rates.

The result at this point is the output of our tunable IF filter. The next step is to convert the IF signal to a baseband signal. The output of the conversion to baseband always contains extra signals that we do not want in our output. The last DSP step is to filter out those signals so we are left with our audio. The baseband filter is a short length (20 coefficients) low pass filter to eliminate frequencies that are far removed from audio. The last step is a call to read the console for input that will halt the signal processing and return to the tuning input dialog. Since the interface is a "teletype terminal," it is not what one would want in a real radio, but it works for our experiments. The console input must be a non-blocking call so that the program will continue DSP operations if there was no terminal activity.

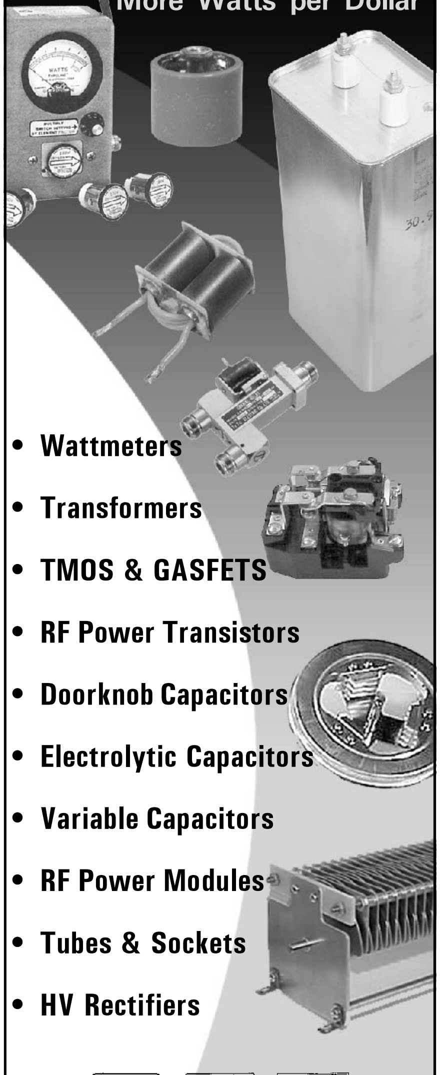
Listing 1 contains the main logic for the DSP loop. The switch statement selects the type of signal to be demodulated. At this point, I have only implemented the AM demodulator. The other modulation types require that a function is implemented, but they all return without performing any action.

Filter Response Program

Gnu Octave has a function (freqz) that is supposed to allow you to plot the frequency of either an Infinite Impulse Response (IIR) or FIR filter. The interface is not especially easy to use, however. The zip file for this installment also includes a frequency response calculator. The output of the program is a set of X-Y points in a text file that you can import into *Gnu Octave*, *Gnuplot*, or a spreadsheet such as *Excel*, so that you can see the plot. Plotting in *Gnuplot* is especially

from MILLIWATTS *to* KILOWATTS℠

More Watts per Dollar℠



- Wattmeters
- Transformers
- TMOS & GASFETS
- RF Power Transistors
- Doorknob Capacitors
- Electrolytic Capacitors
- Variable Capacitors
- RF Power Modules
- Tubes & Sockets
- HV Rectifiers

VISA MasterCard DISCOVER

ORDERS ONLY:
800-RF-PARTS • 800-737-2787

Se Habla Español • We Export

TECH HELP / ORDER / INFO: 760-744-0700

FAX: 760-744-1943 or 888-744-1943

An Address to Remember:
www.rfparts.com

E-mail:
rfp@rfparts.com

RF *Since 1967*

RF RF PARTS™ COMPANY

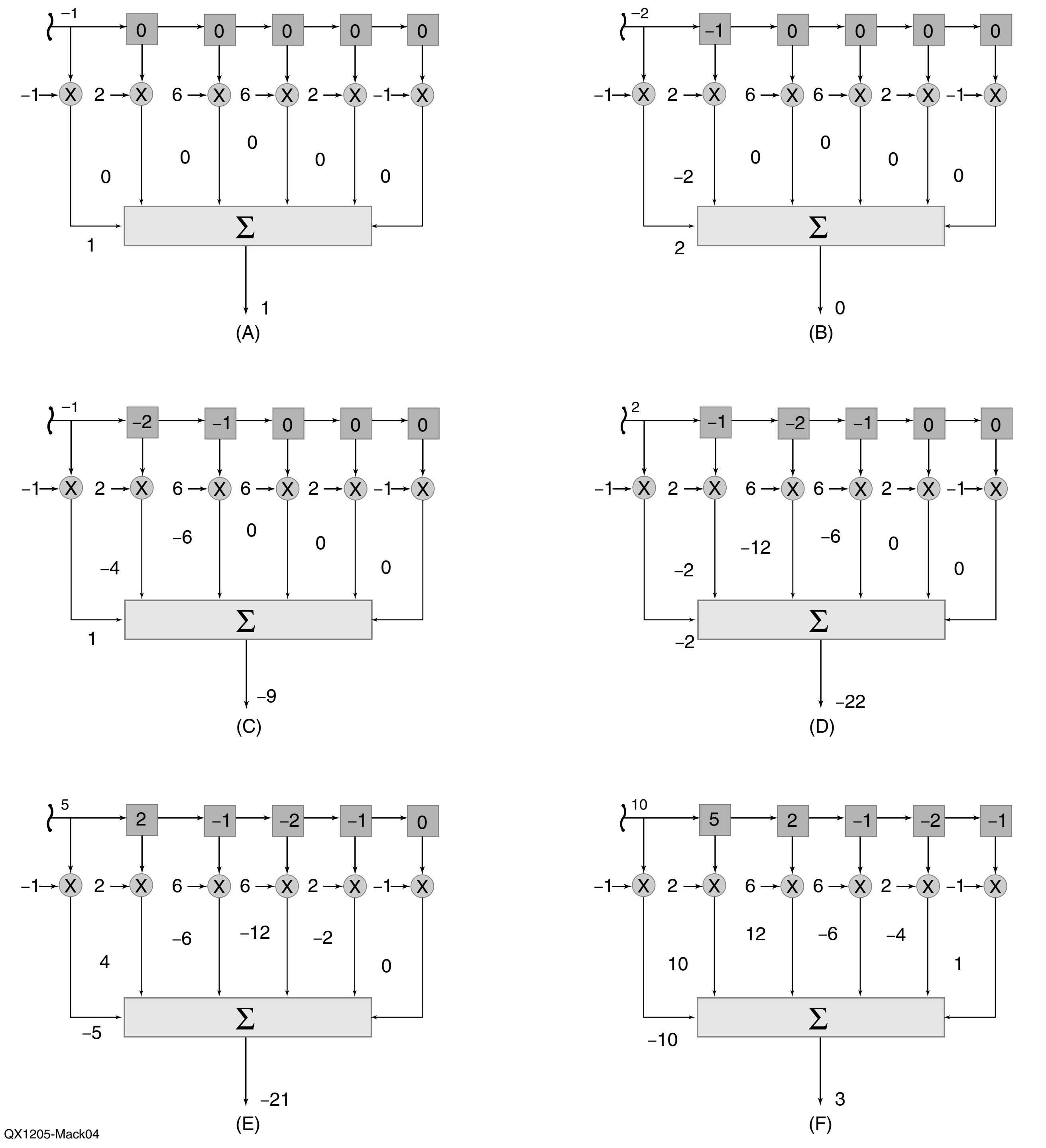


Figure 4 — Here is a graphical description of an FIR filter implementation. The sequence shows how the first 6 samples enter the delay line, and gives the first 6 output samples.

easy, since you can run a script and simply repeat the last command to plot the new data. While we are experimenting with various types of filters and windows, the ability to see the response is very useful. It is especially useful to zoom the plot to just a portion of the total frequency range of the system.

Notes

[1]Hayward, Campbell, Larkin, *Experimental Methods in RF Design*, The American Radio Relay League, 2003.

[2]The software files described in this Column are available for download from the ARRL *QEX* files website. Go to **www.arrl.org/qex-files** and look for the file **5x10_Mack_SDR.zip**.

[3]Press, Teukolsky, Verling, Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.