Selecting FPGAs for FIR filter implementation

By Gordon Hands

Designers face many decisions when implementing a Finite Impulse Response (FIR) filter within an FPGA design. Which parameters are important? What is the best way to approach the design? How should this design be implemented within an FPGA? But the challenge is not as daunting as it sounds. Because FIR filters are among the most common functions implemented within FPGAs, plenty of low-cost IP cores and tools are available to facilitate the design implementation.

Important FIR filter parameters

In their simplest form, FIR filters generate their outputs y(n) by averaging the last N inputs of sample data, x(n). Each sample stored is called a *tap*. Most designs are a little more complicated than this; to provide optimal filter characteristics, it is common to multiply each tap by a coefficient or weighting value, h(i). Figure 1 illustrates a typical FIR filter structure (fully parallel).

The key parameters for a FIR filter are the passband, stopband, stopband attenuation, and passband ripple. Figure 2 shows these parameters for a low-pass filter. For some applications, stopband ripple also may be important and specified. However, stopband attenuation is adequate for most applications. An input sampling rate and desired output sampling rate will be included, along with input and output data resolutions.

Ultimately, the application defines many of the design specifications, though some guidelines can help designers narrow down the possibilities and evaluate design trade-offs. Generally, the closer the passband is to the stopband, the more challenging the design will be. Similarly, specifying very low ripple in the passband or very high stopband attenuation greatly increases the design's complexity. In both these instances, the complexity increases because it requires the design to more closely approximate an ideal filter, which would have infinite taps and coefficients. One thing to keep in mind when specifying ripples and attenuations is data resolution. There is little point specifying attenuation less than the least significant bit of the input data. Likewise, it is not necessary to specify a passband ripple that signifies less than 1 bit at full scale (Figure 3).

The ratio of input to output sample rate also impacts the design. In simple designs, the input and output rates will be equal. However, in many low-pass filter designs, running the output at the same rate as the input is wasteful because high output rate is not required to transmit the information. Here, a decimating FIR is appropriate. For other

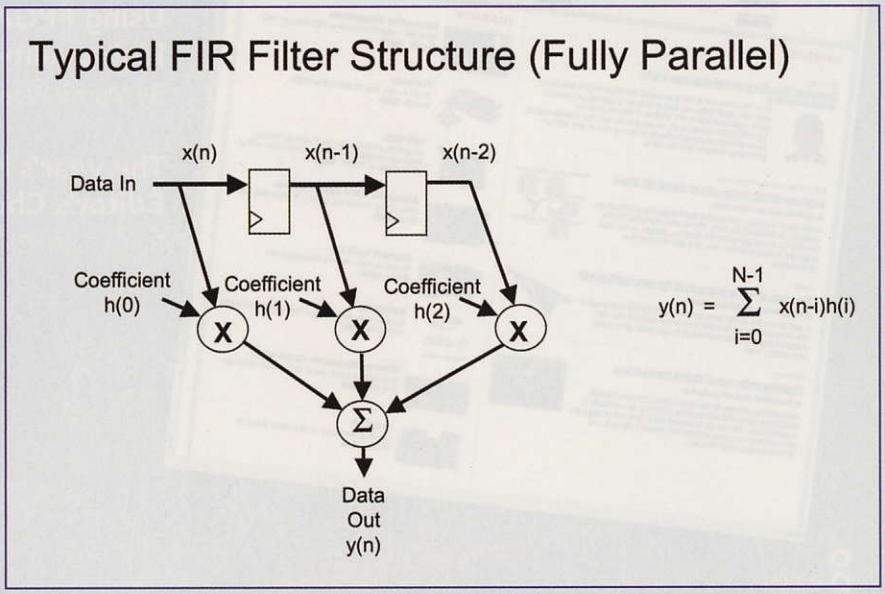

Figure 1

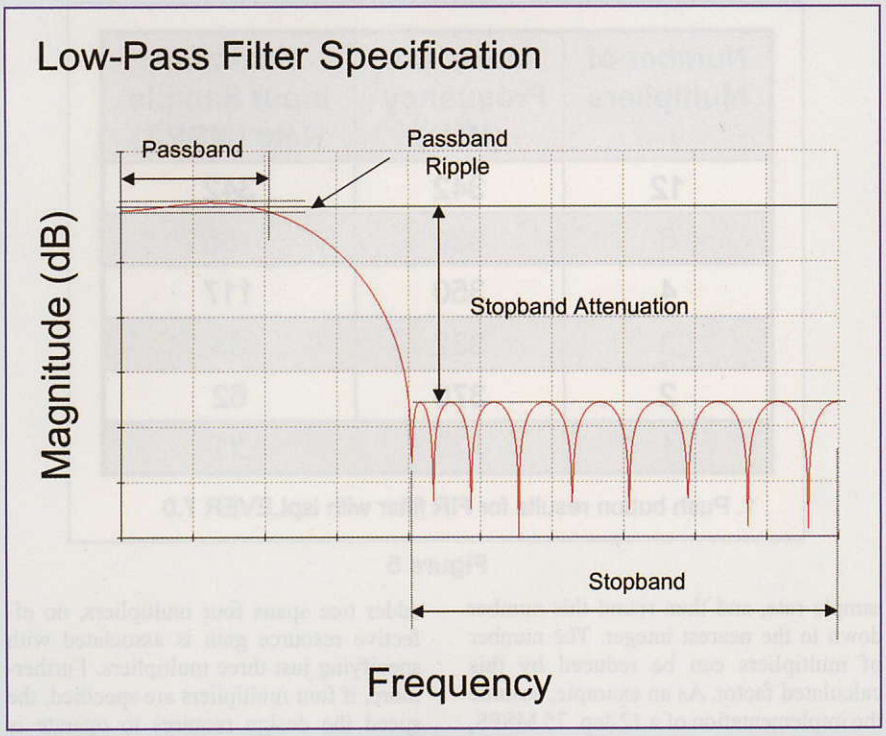

Figure 2

Data Resolution (N)	Maximum Stopband Attenuation 1/2^N (dB)	Minimum Passband Ripple 1/N (dB)
8	-48	0.1880
10	-60	0.0588
12	-72	0.0176
14	-84	0.0051
18	-108	0.0004
20	-120	0.0001

Figure 3

designs that require a higher output rate, an interpolating FIR is appropriate.

Design methodologies

The two common techniques for choosing FIR filter coefficients are the Parks-McClellan method, which implements the Remez algorithm, and windowing the impulse response. The windowing method relies on the fact that the discrete inverse Fourier transform of the desired frequency response represents the FIR filter coefficients. An idealized frequency response is used as the starting point, and then a window function is applied to reduce the number of coefficients and shape the frequency response. Common windowing functions include Rectangular, Triangle, Hanning, Hamming, Kaiser, and Blackman.

Designers do not need to dig too deeply into the mathematical details, as several free, low-cost tools can help with the design. If access to MATLAB is available, a variety of tools from The MathWorks and others can allow FIR design. Otherwise, an Internet search for FIR filter design tools will reveal a number of free, low-cost alternatives.

The first decision designers must make is which design approach to take. Most designers go the Parks-McClellan route because the design inputs are the filter requirements, and this approach generally meets the filter requirements with the lowest number of taps. Using this approach, designers can simply enter the requirements into their chosen tool. Most design tools will then provide the frequency response, impulse response, and filter coefficients. Alternatively, using the windowing approach requires a little more experimentation. Typically, it is necessary to enter the cut-off frequency, windowing approach, and number of taps. The tool will then give the frequency response. If the frequency response is not as desired, the input parameters can be changed and the process iterated.

After selecting and using the design tool, generating the coefficients, and evaluating the frequency response, designers face additional considerations before they can implement the design in an FPGA. Designers must ensure that the frequency response recognizes that the data does not have infinite precision. Most design tools allow designers to define a quantization level and review the results. Often there is not much change with reasonable levels of precision. However, it is much better to discover this now rather than in the lab. Another consideration is exploring other FIR filter designs, such as the Lth band (or Nyquist) filter that has approximately half its coefficients set to zero, reducing mathematical complexity.

Implementing the design in an FPGA

Once designers define the coefficients, they must decide how to implement the design within an FPGA. The first decision is choosing the method of implementing the multiplier elements and determining approximately how fast

Designers must ensure that the frequency response recognizes that the data does not have infinite precision.

they will run. Typically, the two options are implementing the multipliers with the Look-Up Tables (LUTs) used to implement logic within the FPGA or using dedicated DSP elements available within the FPGA. To illustrate these choices, consider the LatticeECP2 FPGA family. An 18 x 18 multiplier implemented using the sysDSP blocks in this FPGA family (Figure 4) consumes no LUTs and will run at above 400 MHz. Using LUTs for the same function yields a performance of approximately 100 MHz and consumes LUTs. Given the dedicated DSP blocks' low LUT usage and high speed, designers will usually choose this approach unless the DSP blocks need to be reserved for some other use.

Number of Multipliers	Maximum Frequency[1] (MHz)	Maximum Input Sample Rate (MSPS)
12	342	342
6	330	165
4	350	117
3	332	83
2	370	62
1	370	31

1. Push button results for FIR filter with ispLEVER 7.0

Figure 5

The next step is to determine the number of physical multipliers that should be used to multiply each tap with its associated coefficient. This step involves a device resource versus speed trade-off. For fastest speed, the number of multipliers will be equal to the number of taps in the FIR filter; this type of filter is a fully parallel implementation. However, in many cases this level of performance is not needed, allowing multipliers to be time-shared.

A quick calculation reveals the number of multipliers to be used. To take a first cut at this, designers can divide the speed that they think moderately complex designs can run in the chosen FPGA by the sample rate, and then round this number down to the nearest integer. The number of multipliers can be reduced by this calculated factor. As an example, assume the implementation of a 12-tap, 75 MSPS, 12-bit filter in the LatticeECP2 FPGA. In this device, 300 MHz filter designs are relatively easy to implement, giving a reduction factor of four and a need for three multipliers.

Before making a final decision, it is useful to consider the ultimate implementation. For instance, the vendor-supplied FIR generator in this design tends to implement FIR filters using the adder tree contained within the sysDSP block. Because this adder tree spans four multipliers, no effective resource gain is associated with specifying just three multipliers. Furthermore, if four multipliers are specified, the speed the design requires to operate is somewhat reduced (Figure 5).

Many FPGA vendors provide automatic FIR generator tools. The final step is to plug the coefficients into the tool and specify the number of multipliers to be used. After generating the filter, timing analysis should be conducted to determine whether the speed assumption is correct. In the example used, the tool reported a 350 MHz maximum speed, well above the assumption of 300 MHz. If the timing target is not met, the assumptions should be adjusted and the design regenerated.

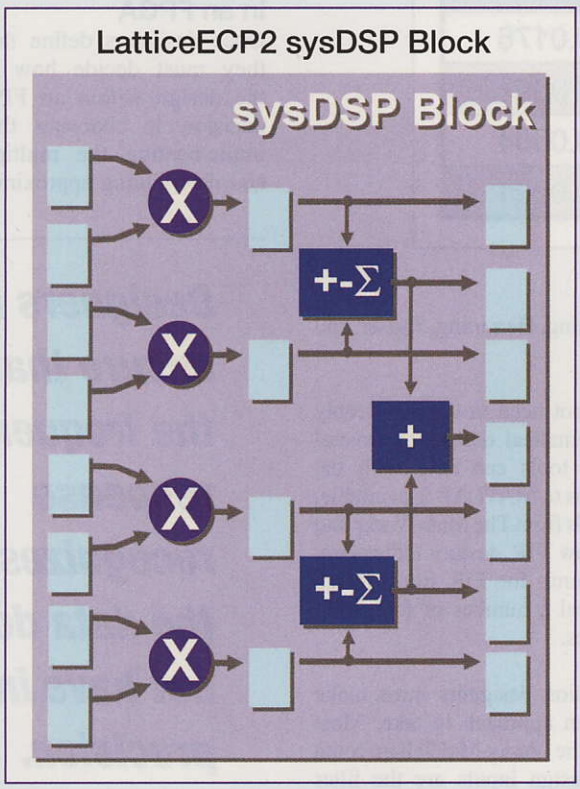

Figure 4

DSP-FPGA.com

Gordon Hands *is director of strategic marketing for Lattice Semiconductor based at the company's Silicon Valley office, where he participates in the definition of Lattice's next-generation programmable device technologies and associated IP cores. Most recently, Gordon has been involved in the definition and launch of the LatticeXP2 nonvolatile FPGA family, LatticeECP2M FPGAs, and the LatticeMico32 embedded microprocessor. Prior to joining Lattice, he worked in system design for four years. Gordon has a B.Eng. from the University of Birmingham, England, and an MBA from Arizona State University.*

Lattice Semiconductor
408-826-6000
gordon.hands@latticesemi.com
www.latticesemi.com