How to Build an Efficient FIR Filter Using System Generator

Xilinx System Generator 2.2 software enables high-level modeling and implementation of DSP systems in Xilinx Platform FPGAs to outperform traditional DSP processors.

XILINX SYSTEM GENERATOR™ For DSP

by Jim Hwang, Senior Manager
DSP Software and Design Methodologies
Xilinx, Inc.
jim.hwang@xilinx.com

Platform FPGAs have become key components for implementing high-performance digital signal processing (DSP) systems, especially in digital communications, video, and image processing applications. FPGAs have memory bandwidth that far exceeds that of microprocessors and DSP processors running at clock rates two to ten times faster, and unlike processors, Platform FPGAs possess the ability to implement highly parallel custom signal processing architectures.

One of the main impediments to wider adoption of FPGAs for signal processing has been the relative unfamiliarity with FPGA technology within the DSP community. What is needed are tools that speak the language of signal processing engineers – for example, MATLAB™ language and Simulink™ design tools from The MathWorks, Inc. – while also supporting FPGA designers who already know how to achieve the highest performance circuit using the least number of FPGA resources.

The new Xilinx System Generator 2.2 software meets the needs of both DSP engineers and FPGA designers. System Generator enables high-level modeling and implementation of DSP systems in the Virtex-II Pro™, Virtex™-II, Virtex, and Spartan™-II families of FPGAs. New in the 2.2 release is an increased capability to generate hardware that approaches the efficiency of hand-crafted modules, both in terms of performance and resource usage. In this article, we describe how System Generator can be used to create a custom FIR (finite impulse response) filter, an operation that lies at the heart of most signal processing systems. We demonstrate that in addition to accessing high-level Xilinx LogiCORE™ modules, you can use your own FPGA knowledge to advantage in customizing a filter data path to minimize resources while achieving high performance. You will see how System Generator provides a wide range of options for implementing signal processing systems.

Xilinx System Generator

The Xilinx System Generator is a state-of-the-art "on-ramp" that allows you to move a DSP algorithm into a Xilinx FPGA quickly and easily. System Generator extends the capabilities of the Simulink system-level simulation environment with bit and cycle-true modeling of an FPGA circuit. System Generator simultaneously provides access to key features in the FPGA fabric, including SRL16E shift register logic, distributed and block memory, and embedded multipliers.

System Generator includes a Simulink library of functional blocks for building DSP, arithmetic, and digital logic circuits. These polymorphic blocks compute their output types based on their inputs, although alternatively, you can specify their quantized output types explicitly. You can combine Xilinx blocks with MATLAB and Simulink blocks to create a realistic test bench and to analyze data computed by your model. The high level of abstraction provided by System Generator greatly simplifies algorithm development and verification.

In addition to a system-level modeling library, System Generator includes a code generator that automatically generates a synthesizable VHDL netlist from your Simulink model. This netlist includes IP (intellectual property) blocks that have been carefully designed for high performance and density in Xilinx FPGAs. System Generator also creates project and constraint files to assist implementation using the Xilinx Foundation™ ISE 4.2i and soon-to-be-released ISE 5.1i tools, as well as the major synthesis tools.

Building an FIR Filter

As a simple but instructive example, let's consider how System Generator can be used to create a parametric finite impulse response (FIR) filter. An N-tap FIR filter is defined by its impulse response, a length N sequence of filter coefficients: $h_0, h_1, \ldots, h_{N-1}$. If $x_0, x_1, x_2, \ldots$, is a sequence of input values, where by convention, we define $x_i = 0$ for $i < 0$, the filter output sequence $y_0, y_1, y_2, \ldots$ is defined by the convolution sum

$$y_n = \sum_{i=0}^{N-1} h_i x_{n-i}$$

That is, the filter output at time n is computed by accumulating a sum of products of the filter coefficients with the N most recent input samples.

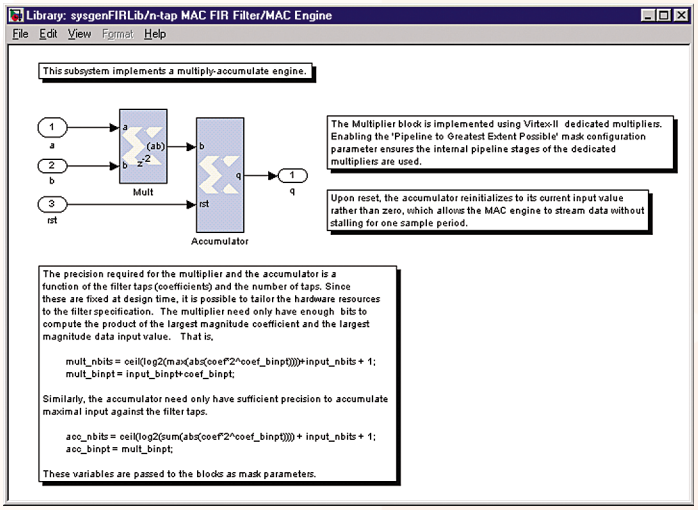

Figure 1 - Multiply-accumulate engine

In practice, all numbers must be represented with a finite number of bits. With a traditional processor, numeric data are typically represented as 8, 16, or 32-bit integers, or in a floating-point representation. In contrast, in an FPGA, we have no such word length limitations. We can create a custom data path processor having an arithmetic precision tailored to the application. System Generator supports this capability by providing an arbitrary precision fixed-point data type. Each block allows us to specify its output precision and the policy for handling quantization and overflow. We can model the system in the Simulink environment under a number of scenarios, and analyze the data to ensure exactly the right precision for the application.

Mapping the Algorithm onto an FPGA

You can implement an FIR filter in an FPGA in many ways. A versatile approach that maps well onto an FPGA employs a multiply-accumulate (MAC) engine to compute the sum of products. As shown in Figure 1, a MAC unit is easily constructed in System Generator using the multiplier and accumulator blocks. The multiplier can be implemented either in the logic fabric or, for Virtex-II family FPGAs, using dedicated 18-bit x18-bit embedded multipliers. System Generator ensures the underlying IP core provides an efficient implementation. Note that upon reset, the accumulator reinitializes to its current input value rather than zero, to avoid a one-clock cycle stall.

Customizing the Data Path

Although Simulink provides a graphical block editor, System Generator should not be mistaken for a "schematic capture" tool. System Generator models are fully customizable and executable in Simulink without recompilation. (In fact, some blocks can be reconfigured during simulation.) Xilinx blocks support Simulink's data type propagation capability, and provide extensive error checking on their parameters and usage. Because System Generator is seamlessly integrated with the Simulink tool suite, you can customize Xilinx blocks in ways that are impossible in schematic and other visual tools.

For example, in our System Generator model we can specify the arithmetic precision of the blocks in the data path using MATLAB expressions, making it possible to minimize the hardware used, and still avoid the possibility of overflow. For a filter with k-bit coefficients and m-bit input, we know that the output

$$|y_n| = \left|\sum_{i=0}^{N-1} h_i x_{n-i}\right| \le \sum_{i=0}^{N-1} |h_i x_{n-i}|$$
$$\le \sum_{i=0}^{N-1} |h_i 2^m| = 2^m \sum_{i=0}^{N-1} |h_i|$$

so the accumulator requires no more than $m + \lceil \log_2 \sum |h_i| \rceil$ bits. This is in practice considerably fewer than the $m + k + \lceil \log_2 N \rceil$ bits implied by the input and coefficient precision. The desired accumulator width is, of course, readily expressed in the MATLAB language. (In Figure 1, the binary point in the fixed-point data is also taken into account.) When you know the input values have limited dynamic range, it may be possible to further tighten the bond. The accumulator width we just derived is a function of the input precision and the MATLAB array that stores the filter coefficients, so if you want to change the input precision or change the filter itself, the same model will "right-size" the data path without any modification. The implications of this ability to use the MATLAB interpreter to customize your System Generator model should not be underestimated – it is unrivaled by any other design flow.

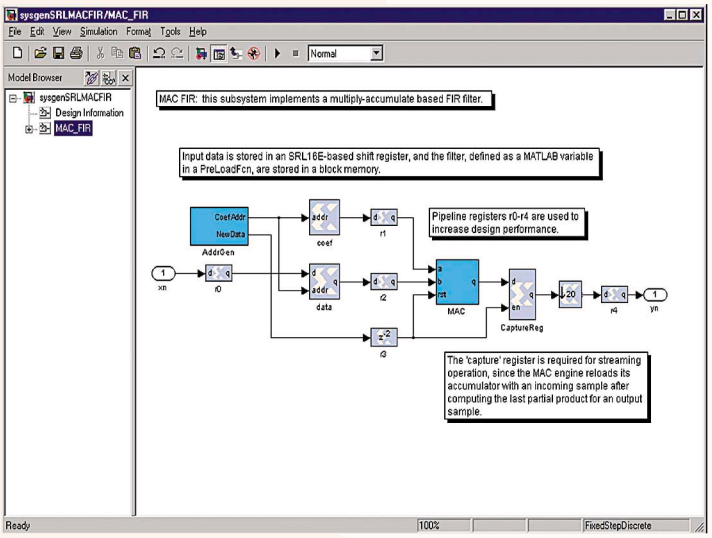

Figure 2 - A multiply-accumulate (MAC)-based FIR filter

Completing the FIR Filter Data Path

As shown in Figure 2, the input data buffer is implemented as an SRL16E-based addressable shift register, and the filter coefficient buffer is implemented as a block memory. (Both are supplied as handcrafted Xilinx LogiCORE™ algorithms.) By storing the filter coefficients in reverse order in memory, the same address counter can be used to drive both buffers.

Because there are N multiply-accumulate operations per input sample, the filter must run internally at N times the data rate to supply a continuous data stream. The capture register on the output of the MAC is used to latch the accumulated sum of products, and its output is down-sampled by N to match the input data rate. This simple filter architecture is quite compact and efficient. A 64-tap non-symmetric filter with 12-bit coefficients and data requires only 110 slices in a Virtex-II XC2V250-6 FPGA, and it runs at 195 MHz, or 3 Msps (ISE 4.2i, production speeds files 1.96).

System Generator provides many ways to tailor the implementation of a design, and changes are tracked automatically and transparently. In the FIR filter, you might choose to use dedicated embedded multipliers for the MAC engine, coupled with a block memory (BRAM) for the coefficient data. (It is natural to do so, because these resources are juxtaposed in the FPGA – although both can also be implemented efficiently in the logic fabric.) Filter throughput can be increased significantly by employing additional MAC engines. System Generator makes this a straightforward extension of our example. When you switch the implementation strategy, latencies are automatically adjusted as necessary in the System Generator model to match the hardware behavior.

Other Implementation Options

Of course, you do not have to build filters from scratch. There is an FIR filter block in the System Generator library that employs distributed arithmetic (DA) to map the computation into the FPGA. This is often the best way to implement a filter, because the underlying IP core supports many efficient architectures, including fully parallel DA, fully serial DA, half band, polyphase interpolators and decimators, and more.

At the same time, there are occasions when you will want to design a custom filter, for example, to exploit symmetries, take advantage of the embedded Virtex-II multipliers, or to build adaptive filters. While providing a high-level simulation and modeling capability, System Generator also allows you to apply your knowledge of the FPGA to map your algorithm onto specific resources. As we saw in our example, you can often parameterize the design using MATLAB functions, so that changing parameters automatically gives you an appropriately customized implementation.

Conclusion

With Virtex/Virtex-II family FPGAs, handcrafted IP LogiCORE algorithms, and System Generator software, Xilinx is rapidly changing the way people think about DSP. Designing a custom DSP data path processor in an FPGA has never been easier.

This article barely scratches the surface of the capabilities of the Xilinx System Generator. The System Generator 2.2 release includes a number of useful annotated demonstration designs, including a QAM demodulator, concatenated codec for digital video broadcast, discrete wavelet transform, and several extensions of the FIR filter discussed in this article, to name but a few. These demonstration designs can be used as-is, or provide a starting point for you to create your own applications. A full-featured, free, 90-day evaluation version of System Generator 2.2 is available for download from the Xilinx website at *www.xilinx.com/systemgenerator_dsp.*

For more information on MATLAB and Simulink software, visit *www.mathworks.com.* Σ