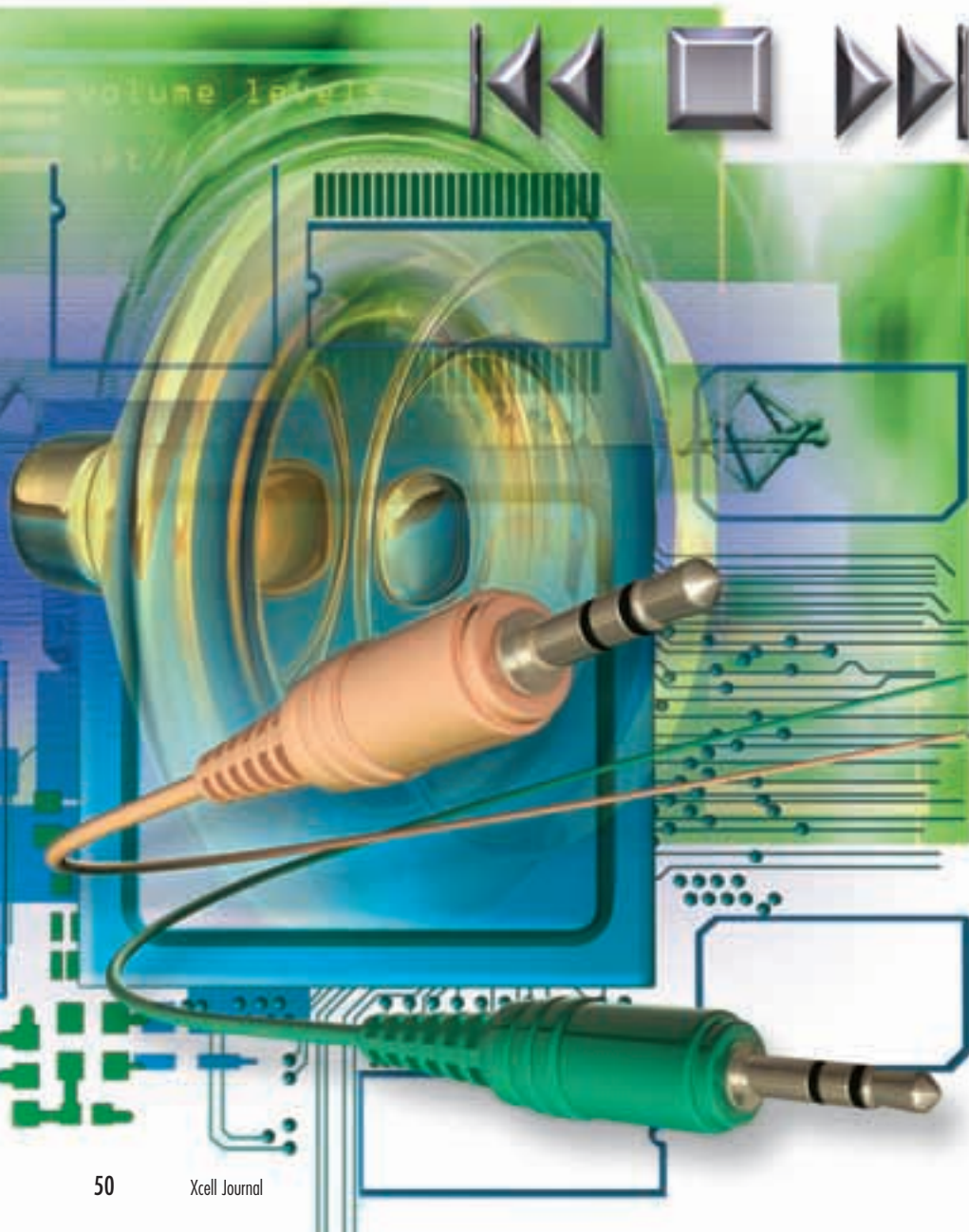


Audio Sample Rate Conversion in FPGAs

An efficient implementation of audio algorithms in programmable logic.



by Philipp Jacobsohn
Field Applications Engineer
Synplicity Deutschland GmbH
philipp@synplicity.com

Derek Palmer
DSP Marketing Manager
Xilinx, Inc.
derek.palmer@xilinx.com

Today, even low-cost FPGAs provide far more computing power than DSPs. Current FPGAs have dedicated multipliers and even DSP multiply/accumulate (MAC) blocks that enable signals to be processed with clock speeds in excess of 550 MHz.

Until now, however, these capabilities were rarely needed in audio signal processing. A serial implementation of an audio algorithm working in the kilohertz range uses exactly the same resources required for processing signals in the three-digit megahertz range.

Consequently, programmable logic components such as PLDs or FPGAs are rarely used for processing low-frequency signals. After all, the parallel processing of mathematical operations in hardware is of no benefit when compared to an implementation based on classical DSPs; the sampling rates are so low that most serial DSP implementations are more than adequate. In fact, audio applications are characterized by such a high number of multiplications that they previously could

only be implemented using very large FPGAs. So audio applications with low sampling frequencies were implemented more efficiently using a DSP than a large FPGA – at a lower cost and with proven software support.

More recently, Synplify DSP, a synthesis tool from Synplicity, allows you to efficiently map even algorithms with large numbers of multiplications and a low sampling rate onto specialized DSP blocks in FPGAs. The tool is based on the popular MATLAB and Simulink tools from The MathWorks.

Algorithms are defined using a special block set or description in the proprietary “M” scripting language and are later translated into an RTL hardware description language. The block set allows both single-rate and multi-rate implementations. It not only generates VHDL and Verilog code but also handles tasks such as fixed-point quantization, pipelining, loop unrolling, and connects to block sets from the Simulink development environment for simulation (see Figure 1).

Application Example: Sampling Rate Conversion

Let’s use a sampling rate converter for audio frequencies as a practical example. This converter can convert a signal from one sampling rate to another with minimal impact on the signal. Such converters are required to process signals with differing sampling rates.

For example, compact discs are sampled at 44.1 kHz, while digital audio tape is usually sampled at 48 kHz. But with data format conversion, playing the source data with the new sampling rate is not sufficient. Playing compact disc material at the sampling rates used for digital audio tape would cause distortions. Thus, the sampling rate must be converted.

When processing audio signals, many sampling frequencies are used: 44.1 kHz, 48 kHz, 96 kHz, and 192 kHz are common. During conversion, you must take care to maintain signal integrity in the audible range between 0 and 20 kHz. Changing the information contained in the signal should be kept to a minimum to limit degradation in audio quality (Figure 2).

Not surprisingly, the implementation of a sampling rate converter for audio frequencies raises two issues in the FPGA:

1. The algorithm issue:

- Highest possible signal-to-noise ratio
- Minimum possible change in the information carried by the original signal
- Efficient description of the algorithm, as the resources consumed in the FPGA are highly dependent on the quality of the description
- Quantization

2. The implementation issue:

- Logically correct implementation of the algorithm
- FPGA resource constraints
- Speed-optimized implementations
- Latency

The conversion requires a high clock speed because the implementation depends on adequate oversampling of the signal being converted. The difference between the FPGA system clock frequency and the signal frequencies being converted must be correspondingly high.

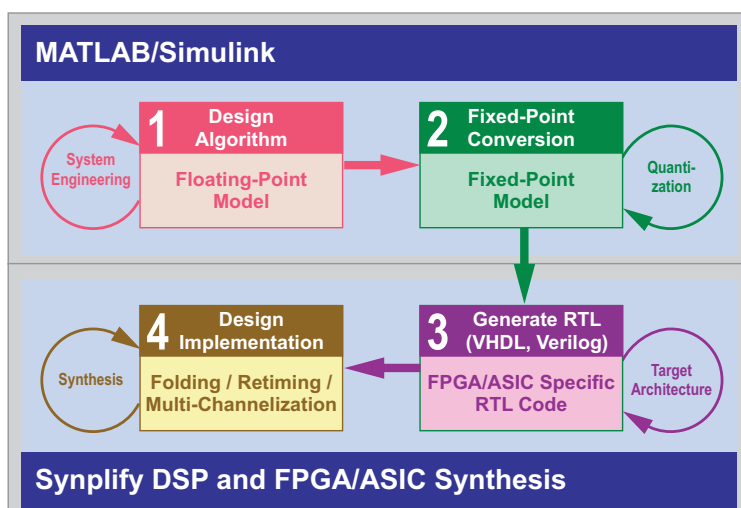


Figure 1 – The model is implemented, quantified, and verified in MATLAB/Simulink. The Synplify DSP tool converts the model into RTL code. The code can be optimized for space or speed.

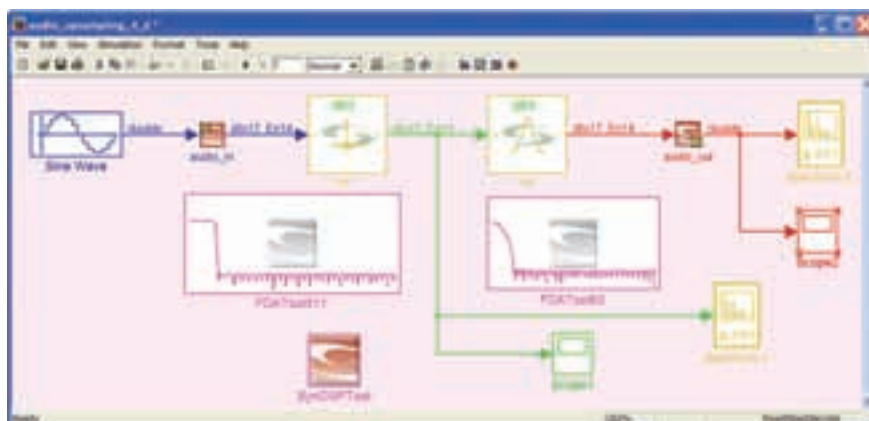


Figure 2 – Modules from the Synplify DSP block set and Simulink FDA tool implement the sampling rate converter. Simulink block set elements perform verification.

The FDA tool helps generate and verify FIR and IIR filters of any kind. It is part of Simulink's signal processing toolbox, which Synplify DSP uses to implement filter structures.

For CD-quality audio signals, the signal-to-noise ratio must also be at least 100 dB. Professional applications even require audio signals of >120 dB. Other low-frequency signals (such as control electronics algorithms) are far less demanding than audio signals when it comes to signal quality.

The Algorithm

A polyphase FIR filter structure converts the sample rates (asynchronous resampling). The algorithm comprises two steps. In the first step, frequencies are oversampled. The second step – linear interpolation – is required to generate a different frequency from a given frequency. The two frequencies are asynchronous to each other.

Resampling the signal in a single step would require far more resources because the filter would be far more complex. This type of implementation would result in several million multiplications. Such a description is inefficient and should be avoided. If a linear interpolation is implemented for the second step, the resulting structures are far simpler (Figure 3).

Efficiently described oversampling (the first step) is the only way to achieve a resource-saving FPGA implementation. The number of computations required will drop dramatically if this part of the circuit is implemented in several cascaded stages rather than in a single computing step.

When implementing the algorithm, you must decide on the target architecture that will perform the computation (DSP or FPGA). Unlike digital signal processors that have a fixed architecture, FPGAs can implement any architecture. They are ultimately limited, however, by the size of the device when implementing large numbers of individual multiplications.

The number of multipliers required increases with the tap of the filter. Each tap results in the use of a DSP block or multiplier. When cascading resampling stages, each filter must perform functions that are

far less complex. In theory, an optimal filter implementation would result from as many individual stages as possible.

The mathematical deduction of how to reduce computing operations has been described extensively in technical literature. Practical results show that while it is necessary to cascade filter stages, the number of cascades must be limited. If you introduce too many cascaded stages, you could exceed the available resources to implement the design. If an FPGA is used

Verilog code. FFT and SCOPE elements from Simulink block sets conduct spectrum analysis and verification of the dynamic response. These blocks are exclusively used for functional verification, including floating-point to fixed-point conversion effects (quantization). The blocks are not implemented in hardware.

The first part of the algorithm implementation comprises two FIR filters: the first has 512 taps and the second has 64 taps. The RTL code resulting from over-

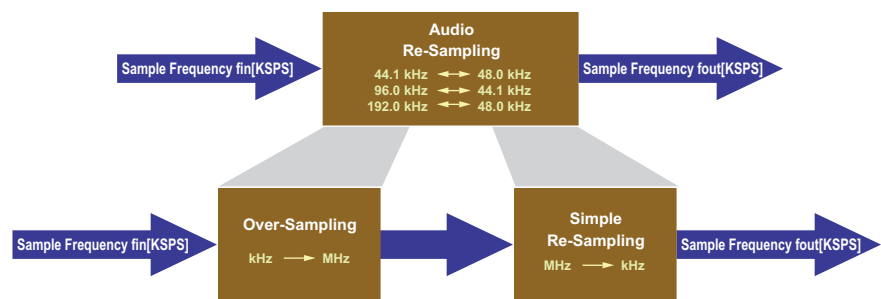


Figure 3 – The sample-rate converter is implemented in two steps (one, oversampling; two, linear interpolation) to improve efficiency.

as the target architecture, two stages has proved to be optimal.

The entire circuit comprises two relatively simple filters for oversampling and a simple linear interpolator. This structure can be efficiently mapped onto an FPGA.

The Implementation

You can implement the circuit in Simulink using the Synplify DSP block set and Simulink's filter design and analysis (FDA) tool. The FDA tool helps generate and verify FIR and IIR filters of any kind. It is part of Simulink's signal processing toolbox, which Synplify DSP uses to implement filter structures.

All circuit components from the Synplify DSP block set or the FDA tool, which are defined between a PortIN and a PortOUT description, generate VHDL or

sampling therefore contains a total of 576 multiplications, which is why using an FPGA does not appear to be commercially viable. Such a large FPGA would be cost-prohibitive, requiring the largest Xilinx® Virtex™-5 XC5VSX95T device with its 640 DSP48 blocks.

All multiplications that are not mapped onto dedicated hardware structures (DSP blocks) must be built from generic logic resources (LUTs or registers). This results in higher resource requirements as well as a lower maximum clock speed. Dedicated DSP48 blocks are far more efficient multipliers than generic logic cells (Figure 4).

Optimization

Synplify DSP's folding option allows you to minimize the number of multipliers used. Those circuits operating at low sampling

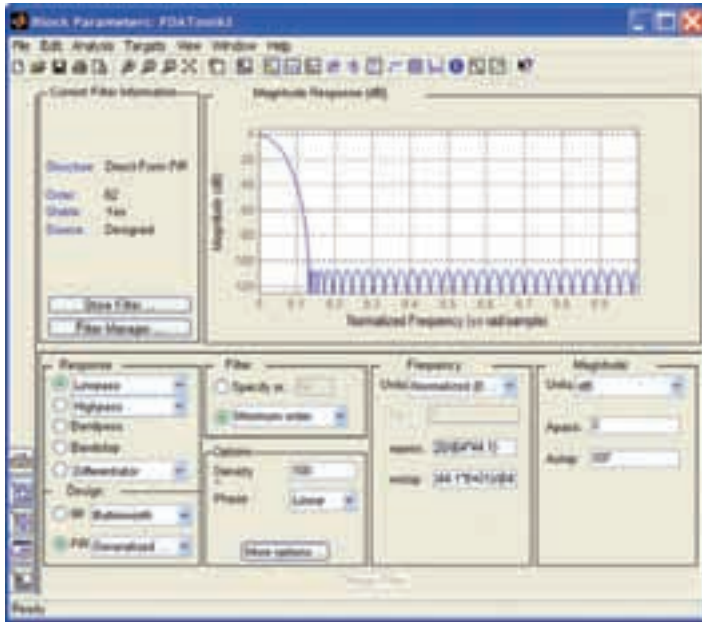


Figure 4 – Implementing filters using Simulink's filter design and analysis (FDA) tool

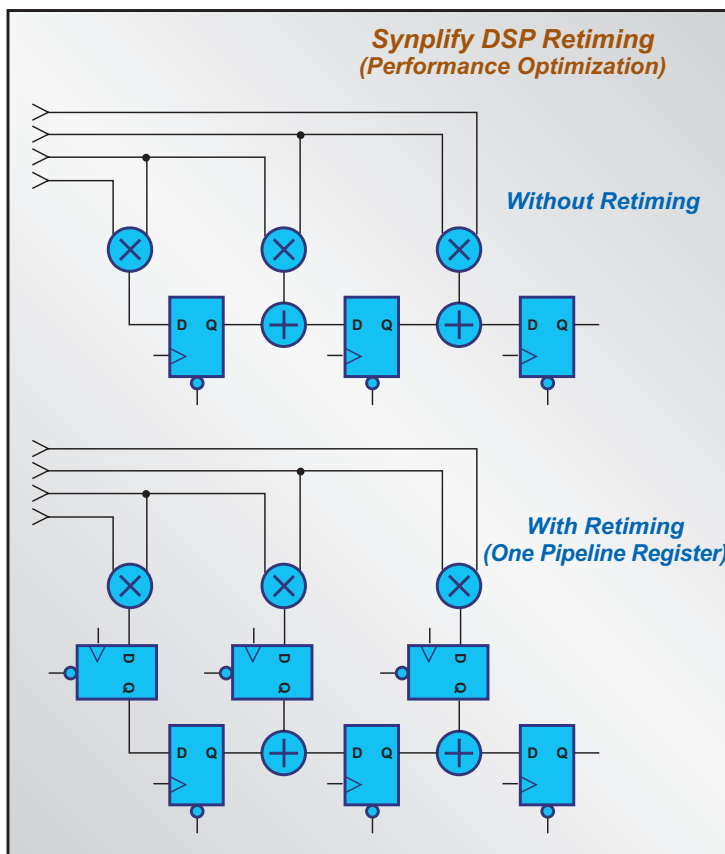


Figure 5 – You can dramatically reduce the FPGA resources required by using the folding feature.

frequencies can particularly benefit from this optimization.

The idea is simple. Normally, one hardware multiplier is used for each multiplication, even when the sampling frequency is in the kilohertz range. However, FPGAs can operate with clock speeds in the triple-digit megahertz range. If the hardware multiplier operates at the system frequency of the FPGA, multiplications can be processed sequentially using a time multiplexing process.

Let's say that the sampling frequency of the circuit is 3 MHz and the FPGA can run at a maximum of 120 MHz. Each hardware multiplier can perform 40 computing operations if the multipliers are run at the system frequency. The necessary hardware is therefore reduced by a factor of 40. This means that a sampling rate converter as described above (or any other circuit using low sampling frequencies) can be "folded" to the point where only very few hardware multipliers are required. Therefore, this converter can also be implemented in the smallest available low-cost FPGA and thus is a real alternative to DSPs.

Of course, it is also possible to offload particularly computationally intensive algorithms from a DSP to an FPGA, thereby reducing processor load. This is particularly useful if your DSP application has exceeded the performance capability and if you have a significant investment in application source code targeted at a specific DSP architecture (Figure 5).

Because the folding feature in Synplify DSP also supports multi-rate systems, you can reduce the number of multipliers required even more than in a system with a single sampling frequency. Oversampling is performed using two FIR filters. These filters run at different sampling frequencies. The filter running at a higher sampling frequency is folded using a folding factor that you specify.

The filter with the lower sampling frequency is folded using a correspondingly higher factor. This factor is obtained by multiplying the difference between the sampling frequencies of the two filters by the folding factor. For example, if the sampling frequency of one filter is 8 times

higher than the sampling frequency of the other filter, the faster filter is folded by a factor of 8 and the slower filter is folded by a factor of 64.

In this way, it is even possible to produce space-optimized circuits running at very high sampling rates that normally cannot be folded. For example, if a system runs at a sampling rate of 200 MHz with a folding factor of 2, the system frequency increases to 400 MHz.

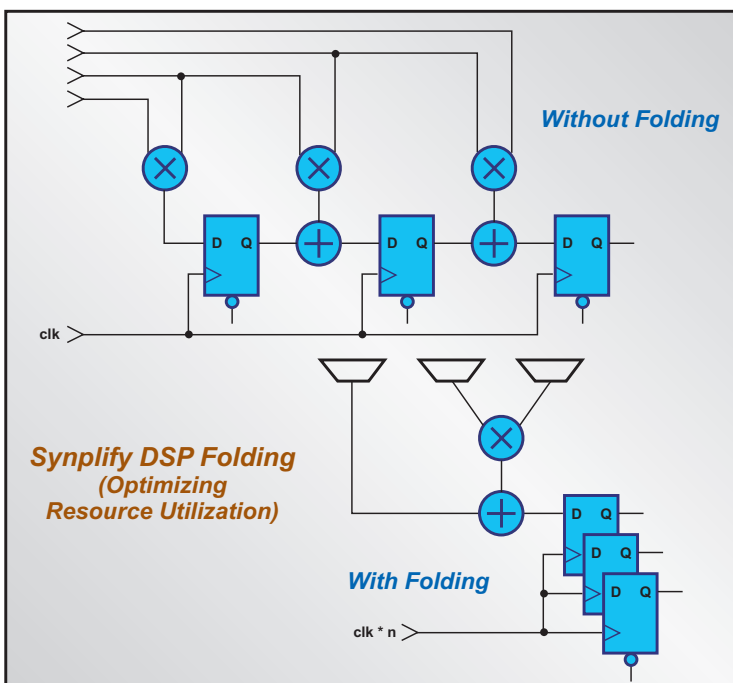


Figure 6 – Using the retiming feature, you can define the maximum latency allowed for the circuit. Synplify DSP then automatically adds pipeline stages until achieving the desired frequency.

Alternatively, you can define a folding factor of 1. Those circuit components running at the highest sampling rate are not folded. However, all circuit components of a multi-rate system running at slower sampling frequencies benefit from folding and space-optimized implementation. You only need to define the folding factor for the system as a whole. Folding is then propagated automatically across all sampling frequencies.

The folding feature can be combined with additional optimization functionality – the retiming feature. If a system does not meet the target frequency requirements, you can add pipeline stages until you

achieve the desired rate. This is particularly important for circuits with high folding factors, which need to operate at a correspondingly high system speed.

You can also use retiming for circuits with little or no folding except where the performance limit of the FPGA is reached. Adding pipeline stages allows the number of combinatorial gates between two registers (logic levels) to be reduced, which increases the system clock speed.

speed can be increased significantly with little effort using registers.

Of course, adding pipeline stages increases system latency. By introducing a retiming factor of 8, for example, the result of the computation will appear eight system clock cycles (not sampling frequency cycles) later at the FPGA’s output. You must take this into account when embedding a circuit in a system (Figure 6).

It is particularly important to ensure that the optimizations described previously do not impact the original MATLAB model described in Simulink. Verification allows the algorithm to be validated and the impact of quantization effects to be represented. The Synplify DSP software block set allows floating- to fixed-point conversion using either truncation (elimination of irrelevant bits), rounding (in case of underflow), or saturation (in case of overflow). As soon as the simulation shows that the algorithm works as intended, the RTL code can be generated. Optimizing the VHDL or Verilog code may change latency, but not the operation of the circuit.

Conclusion

The Synplify DSP tool is based on the industry-standard MATLAB/Simulink software from The MathWorks. A block set provides a library of standard components that you can use to implement complex algorithms. Apart from basic components such as add, gain, and delay, the library contains many complex functions such as FIR or IIR filters and CORDIC algorithms. All features, including the highly complex FFT or Viterbi decoder, can be parameterized as you like. It is also possible to create user-defined libraries or integrate existing VHDL or Verilog code into a Simulink model.

Synplify DSP allows the implementation of both single- and multi-rate systems. Using folding, multi-channelization, or retiming, you can optimize the code for either size or speed. The RTL code generated is always generic, non-encrypted code that is synthesizable using popular tools.

For best results with FPGAs, Synplify recommends its synthesis product, Synplify Pro. An ASIC variant of the development environment is also available now. 