Floating- to fixed-point MATLAB algorithm conversion for FPGAs

Tom Hill, Xilinx - June 04, 2007

In a recent survey conducted by AccelChip Inc. (recently acquired by Xilinx), 53% of the respondents identified floating- to fixed-point conversion as the most difficult aspect of implementing an algorithm on an FPGA (Figure 1).

![Figure 1. AccelChip DSP design challenges survey.](source)

Although MATLAB is a powerful algorithm development tool, many of its benefits are reduced during the fixed-point conversion process. For example, new mathematical errors are introduced into the algorithm because of the reduced precision of the fixed-point arithmetic. You must rewrite code to replace high-level functions and operators with low-level models that reflect the actual hardware macro-architecture. And simulation run times can be as much as 50 times longer. For these reasons, MATLAB, the overwhelming choice for algorithm development, is often abandoned in favor of C/C++ for fixed-point modeling.

**Generating Fixed-Point Models**

The fixed-point representation of a floating-point MATLAB algorithm will not truly reflect the response of the final hardware if the high-level functions and operators are not replaced with hardware-accurate macro-architectures (Figure 2).
\[ y = \frac{a}{b}; \]

![Diagram of divide operation](image)

**Figure 2. Replacing built-in operators and functions.**

This is highlighted in Figure 3, which compares the fixed-point response of a MATLAB divide operator against a hardware-implementable CORDIC divide algorithm using a random set of input vectors quantized to 8-bit signed twos complement. Depending on the data values, a significant divergence exists between the calculated outputs.

![Plot of MATLAB divide vs CORDIC divide](image)

**Figure 3. Fixed-point response of the MATLAB "/" versus CORDIC.**

During the fixed-point generation process, the AccelDSP Synthesis tool's IP Explorer technology will automatically replace high-level MATLAB functions and operators with hardware-accurate representations (Figure 4). This step is transparent and does not require MATLAB code modifications. You can redefine the initial macro- and micro-architecture selections by using a synthesis directive.

Once these operations have been replaced with hardware-accurate macro-architectures, the process of quantization may begin.
Graphically Assisted Auto-Quantization

The FPGA fabric, unlike a fixed-point DSP processor, allows for variable, fixed-point word lengths. By not limiting a variable to a fixed 16- or 24-bit boundary, you can perform arithmetic calculations requiring bit growth without incurring additional numeric error. This is a tremendous advantage for applications such as radar, navigation, and guidance systems that require a high degree of numeric precision.

In most cases bit growth rules are straightforward and well understood. The result of an addition, for example, grows by one bit and the result of a multiplication grows to a length equal to the sum of the input word lengths (Figure 5). Making these determinations for variables in an actual design, however, is a highly iterative process. Allowing unchecked bit growth to occur is expensive in hardware and generally unnecessary. If you're savvy, you can employ a variety of techniques to minimize word lengths while preserving numerical accuracy.

The process of determining an initial quantization value for a variable and the subsequent refinement of that value is well suited for automation. The AccelDSP Synthesis tool includes automated floating- to fixed-point conversion in which the floating-point MATLAB model is analyzed during simulation to determine the dynamic range requirements of the input data and constants. These values provide the starting points to an auto-quantization process that then draws upon a wealth of built-in experience, gained from more than 6,000 designs, to determine optimal word lengths for the downstream variables.

The initial fixed-point model obtained through auto-quantization provides a good starting point, but refinements to the model are generally necessary. This process is highly iterative and tightly coupled to analysis of the data effects. To minimize this iteration cycle time, the AccelDSP Synthesis tool provides an accelerated fixed-point simulation flow.

Analyzing Fixed-Point Data Effects
MATLAB provides a highly efficient environment for developing the mathematics of an algorithm that you can generally accomplish with a small set of simulation vectors. When targeting that algorithm to fixed-point hardware, however, you will need increased data sets to accurately determine the real-world environment response. MATLAB, which is an interpreted simulator, may not provide the necessary performance for these larger, more CPU-intensive fixed-point simulations. For this, developers often turn to C/C++.

**Accelerated Fixed-Point Simulation**

The AccelDSP Synthesis tool’s M2C-Accelerator automatically generates a hardware-accurate fixed-point C++ model and test bench to accelerate fixed-point simulations. Eliminating the manual recoding step saves development time and minimizes the introduction of errors. Because C++ is compiled, it can provide as much as a 1000x simulation performance advantage (Figure 6). This level of performance is often necessary for the large vector sets required to understand fixed-point data effects.

If you wish to continue using the MATLAB visualization environment, including the plotting features, M2C-Accelerator also generates a fixed-point C/C++ dll that can be simulated with the original MATLAB test bench script file.

When you have obtained the initial fixed-point results, the process of analysis and refinement can begin. The AccelDSP Synthesis tool provides a set of graphical aids, including tabulated reports, variable probes, and plots to assist in this process.

![Figure 6. FFT example simulation run times.](image)

**Observing Fixed-Point Bit Growth**

A design must be considered in its entirety to effectively convert a floating-point algorithm into a fixed-point model. If left unchecked early in the datapath, bit growth can quickly escalate to produce unreasonable hardware, while overly constrained bit growth may result in an unacceptable loss of numeric accuracy. A common technique to gain better observability into bit-growth progression is to enter the variables into a spreadsheet. The AccelDSP Synthesis tool provides this same level of observability by generating a tabular, formatted Fixed Point Report (Figure 7).

Before optimizing the hardware, you must obtain an acceptable fixed-point response. If the signal-to-noise ratio (SNR) of an output is not above a desired specification, then adjustments to the inferred quantization values are required. This process typically starts by looking for gross errors caused by variable overflows and underflows.
Overflows and Underflows

Poor assumptions about the dynamic range of the input data can lead to large fixed-point errors caused by overflowing the most significant bit (MSB) and (to a lesser degree) underflowing the least significant bit (LSB) of a variable. You will need to address these errors first before observing and correcting more subtle fixed-point errors.

Overflow and underflow reporting, inherent to MATLAB fixed-point data types, are not native to C/C++ and are often sacrificed during the model rewrite. The C++ models generated by M2C-Accelerator, however, include quantization routines that report all overflows and underflows encountered during a simulation. When these conditions occur, they are summarized in the "Verify FixedPoint Report" (Figure 8).

Verify FixedPoint Report

Once you have addressed any overflow and underflow issues, the refinement of the fixed-point model...
becomes more dependent on visualization. If additional fixed-point numeric errors persist, then you must analyze the effects of constants. Otherwise, you can continue the process of refining the hardware by reducing variable bit widths. In both cases, knowing the fixed-point error introduced by the quantization of a particular variable is a valuable aid in the refinement process.

**Fixed-Point Visualization**

Determining the appropriate fixed-point response of an algorithm to a given set of data is generally not an exact science. You will often have to make compromises in numerical accuracy to improve hardware efficiency. This process is highly iterative and tightly coupled to a visual analysis of the fixed-point effects displayed in plots. Observing an unacceptable SNR on an output signal, however, does not always indicate where a quantization value has been incorrectly specified. For that, additional analysis is necessary.

To assist in this process, AccelDSP Synthesis's AccelProbe graphically compares the floating- and fixed-point values for any variable during a given simulation (Figure 9). If you are using AccelProbe, you will quickly gain a sense of the magnitude that a particular variable's contribution makes to the cumulative error of the final result. You can "probe" a variable by adding the statement, "accel_probe(variable_name)" to the MATLAB source.

\[
x = a/b;
\]

\[
\text{accel Probe}(x);
\]
Figure 9. Accelprobe plot for a variable.

The "Fixed-Point Histogram" plot gives you a sense of how often a value may be encountered during simulation. The additional hardware required to store a value in the upper or lower dynamic range may be of little value if that value rarely occurs.

**Conclusion**
When inventing the mathematics of a DSP algorithm, MATLAB is the natural choice and should be used unencumbered by hardware considerations. Converting an algorithm into a fixed-point model for implementation on an FPGA is an involved process that benefits greatly from the automation, acceleration, and visualization offered by the AccelDSP Synthesis tool.

For more information about the AccelDSP Synthesis tool, visit [www.xilinx.com/dsp](http://www.xilinx.com/dsp).

**About the author:**
Tom Hill is the Technical Marketing Engineer for DSP Tools at Xilinx, Inc. He can be reached at tom.hill@xilinx.com

[Editor's Note: This article first appeared in the Xilinx DSP Magazine and is presented here with the kind permission of Xcell Publications.]