



WP239 (v1.0) April 19, 2006

AccelDSP Synthesis Tool Floating-Point to Fixed-Point Conversion of MATLAB Algorithms Targeting FPGAs

By: Thomas Hill

In a recent survey conducted by AccelChip, Inc. (recently acquired by Xilinx), 53 percent of the respondents identified floating-point to fixed-point conversion as the most difficult aspect of implementing an algorithm on an FPGA. Many of the benefits of MATLAB that make it such a powerful algorithm development tool are reduced during the fixed-point conversion process. New mathematical errors are introduced into the algorithm due to the reduced-precision of the fixed-point arithmetic. Code must be rewritten to replace the use of high-level functions and operators with low-level models that reflect the actual hardware macro-architecture while simulation run times can take up to 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.

© 2006 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Generating Fixed-Point Models

Replacing High-Level Functions and Operators with Hardware Accurate Models

The fixed-point representation of a floating point MATLAB algorithm does not truly reflect the response of the final hardware if the high-level functions and operators are not replaced with hardware accurate macro-architectures. See [Figure 1](#).

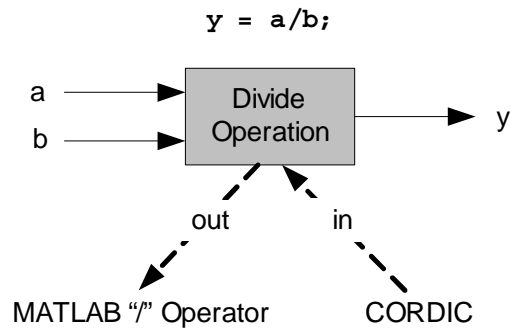


Figure 1: Replacing Built-in Operators and Functions

This is highlighted in [Figure 2](#) which compares the fixed-point response of the MATLAB divide operator against a hardware implementable CORDIC divide algorithm using a random set of input vectors quantized to 8-bit signed two's complement. Depending on the data values, there can be a significant divergence between the calculated outputs.

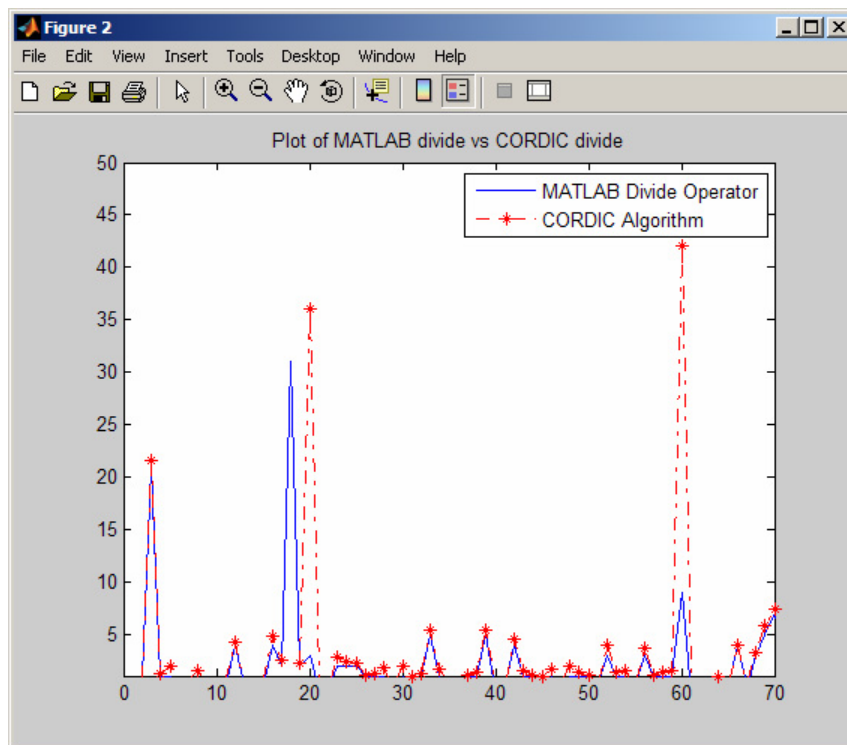


Figure 2: Fixed-Point Response of the MATLAB “/” vs. CORDIC

During the fixed-point generation process, AccelDSP IP Explorer™ technology automatically replaces high-level MATLAB functions and operators with hardware accurate representations (Figure 3). This step is transparent to the user and does not require MATLAB code modifications. The initial macro and micro architecture selections can be redefined using a synthesis directive.

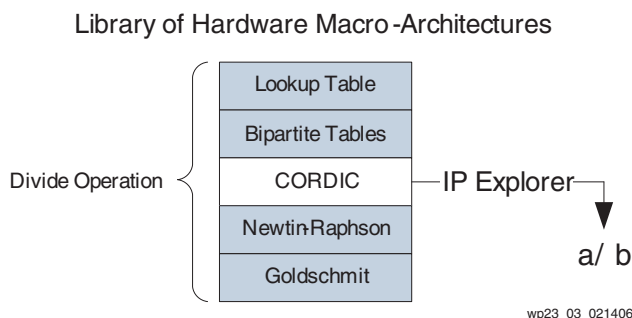


Figure 3: Automatic Hardware Accurate IP Insertion

After these operations have been replaced with hardware accurate macro-architectures, the process of quantization can begin.

Graphically Assisted Auto-Quantization

The FPGA fabric, unlike a fixed-point DSP processor, allows for variable length fixed-point word lengths. By not limiting a variable to a fixed 16- or 24-bit boundary, arithmetic calculations requiring bit growth can be performed without incurring additional numeric error. This is a tremendous advantage for applications such as radar, navigation, and guidance systems, but comes at the cost of increased hardware.

In most cases, bit growth rules are straightforward and well understood (Figure 4). For example, the result of an addition grows by one bit and the result of a multiplication grows to a length equal to the sum of the input word lengths. However, making these determinations for variables in an actual design is a highly iterative process. Allowing unchecked bit growth to occur is expensive in hardware and generally unnecessary. The savvy designer can employ a variety of techniques to minimize word lengths while preserving numerical accuracy.

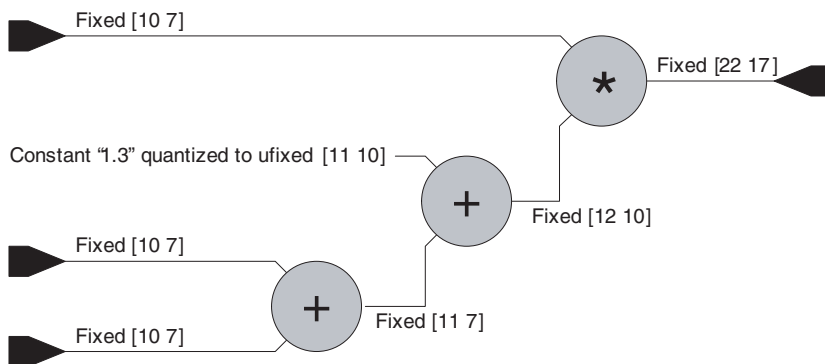


Figure 4: Fixed-Point Bit Growth

The process of determining an initial quantization value for a variable and the subsequent refinement of that value is well suited for automation. AccelDSP™ Synthesis Tool includes automated floating-point to fixed-point conversion, where 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 over 6000 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 this model are generally necessary. This process is highly iterative and tightly coupled to analysis of the data effects. To minimize this iteration cycle time, 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 can generally be done with a small set of simulation vectors. When targeting that algorithm to fixed-point hardware, however, increased data sets are required to accurately determine the real world environment response. MATLAB, which is an interpreted simulator, might 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 M2C-Accelerator automatically generates a hardware accurate, fixed-point C++ model and testbench 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 up to a 1000x simulation performance advantage. This level of performance is often necessary for the large vector sets required to understand the fixed-point data effects.

For users who wish to maintain the use of 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 testbench script file.

Figure 5 shows a comparison of simulation run times.

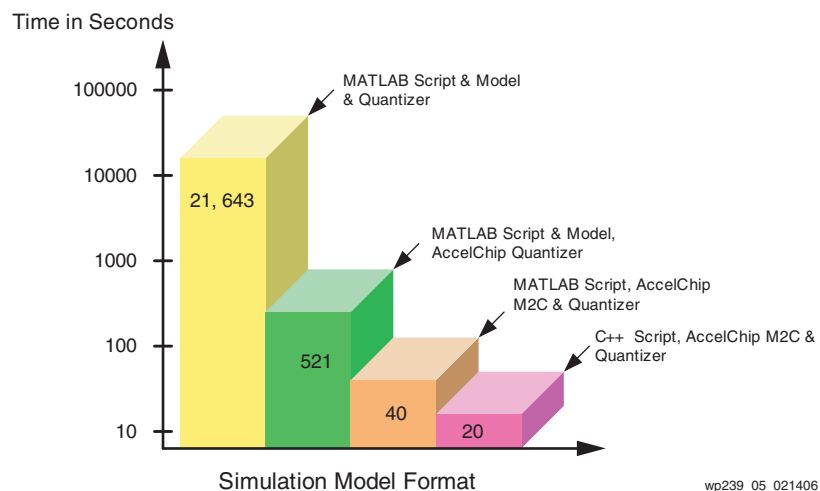


Figure 5: FFT Example Simulation Run Times

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

Observation of Fixed-Point Bit Growth

A design must be considered in its entirety to effectively convert a floating-point algorithm into a fixed-point model. Bit growth, if left unchecked early in the datapath, can quickly escalate to produce unreasonable hardware, while overly constrained bit growth can result in an unacceptable loss of numeric accuracy. A common technique used by developers to gain better observability into bit-growth progression is to enter the variables into a spreadsheet. AccelDSP provides this same level of observability through the generation of a tabular formatted Fixed Point Report (Figure 6).

Before the hardware can be optimized an acceptable fixed-point response must be obtained. 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.

x1	1 x 1	fixed floor wrap [16 10]	Directive (delete)
x2	1 x 1	fixed floor wrap [16 10]	Directive (delete)
x3	1 x 1	fixed floor wrap [16 10]	Directive (delete)
d	1 x 1	fixed floor wrap [16 10]	Directive (delete)
init_flag	1 x 1	ufixed floor wrap [1 0]	Auto Inferred

Outputs

Name	Shape	Quantizer	Quantizer Source
e	1 x 1	fixed floor wrap [30 21]	Directive (delete)

Variable List

Name	Shape	Quantizer	Quantizer Source
Ci	1 x 1	fixed floor wrap [16 14]	Directive (delete)
M <i>constant</i>	1 x 1	ufixed floor wrap [2 0]	Auto Inferred
R <i>persistent</i>	4 x 3	fixed floor wrap [16 10]	Directive (delete)
col	1 x 1	ufixed floor wrap [3 0]	Auto Inferred
cos_phi	1 x 1	fixed floor wrap [16 14]	Inferred from MATLAB Source
lambda <i>constant</i>	1 x 1	ufixed floor wrap [8 8]	Auto Inferred
p <i>persistent</i>	4 x 1	fixed floor wrap [18 10]	Directive (delete)
px	1 x 1	fixed floor wrap [26 18]	Auto Inferred
py	1 x 1	fixed floor wrap [18 10]	Inferred from Directive
row	1 x 1	ufixed floor wrap [3 0]	Auto Inferred
rx	1 x 1	fixed floor wrap [24 18]	Auto Inferred
ry	1 x 1	fixed floor wrap [16 10]	Directive (delete)

Figure 6: AccelDSP Generate Fixed-Point Report for QRD-RLS Adaptive Filter

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. These errors need to be addressed first before more subtle fixed-point errors can be observed and corrected.

Overflow and underflow reporting, inherent to the MATLAB fixed-point data types, are not native to C/C++ and are often sacrificed during the model rewrite. However, the C++ models generated by M2C-Accelerator include quantization routines that report all overflows and underflows encountered during a simulation. When these conditions occur, they are summarized for the user in the Verify FixedPoint Report shown in Figure 7.

Verify FixedPoint Report

Elapsed Time: 8.44 seconds

- Overflows				
# of Overflows	Location	Quantizer	Overflow Value	
			Min	Max
3	test.m (60)	ufixed wrap floor [5 0]	35.8594	255.996

- Underflows (rounded to zero)				
# of Underflows	Location	Quantizer	Underflow Value	
			Min	Max
46	test.m (60)	ufixed wrap floor [5 0]	0.015625	0.988281
7	mrdivide_001.m (81)	fixed wrap floor [8 7]	0.00111541	0.00621219
1	test_script.m (24)	fixed wrap floor [8 7]	0.00769819	0.00769819
1	test_script.m (25)	fixed wrap floor [8 7]	0.0029996	0.0029996

Figure 7: AccelDSP Verify FixedPoint Report

After overflow and underflow issues have been addressed, the refinement of the fixed-point model becomes more dependent on visualization. If additional fixed-point numeric errors persist, then the effects of constants must be analyzed. Otherwise, the process of refining the hardware by reducing variable bit widths can proceed. In both cases, knowing the fixed-point error introduced by the quantization of a particular variable is a valuable aid in this 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. Compromises are often made in numerical accuracy to improve the hardware efficiency. This process is highly iterative and tightly coupled to the visual analysis of the fixed-point effects displayed in plots. Observing an unacceptable signal-to-noise ratio on an output signal, however, does not always indicate where a quantization value has been incorrectly specified. For that, additional analysis is needed.

To assist in this process, AccelProbe graphically compares the floating-point and fixed-point values for any variable during a given simulation. Developers using AccelProbe quickly gain a sense of the magnitude of the contribution that a particular variable makes to the cumulative error of the final result. A variable can be *probed* by adding a single line of MATLAB as shown in Figure 8.

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

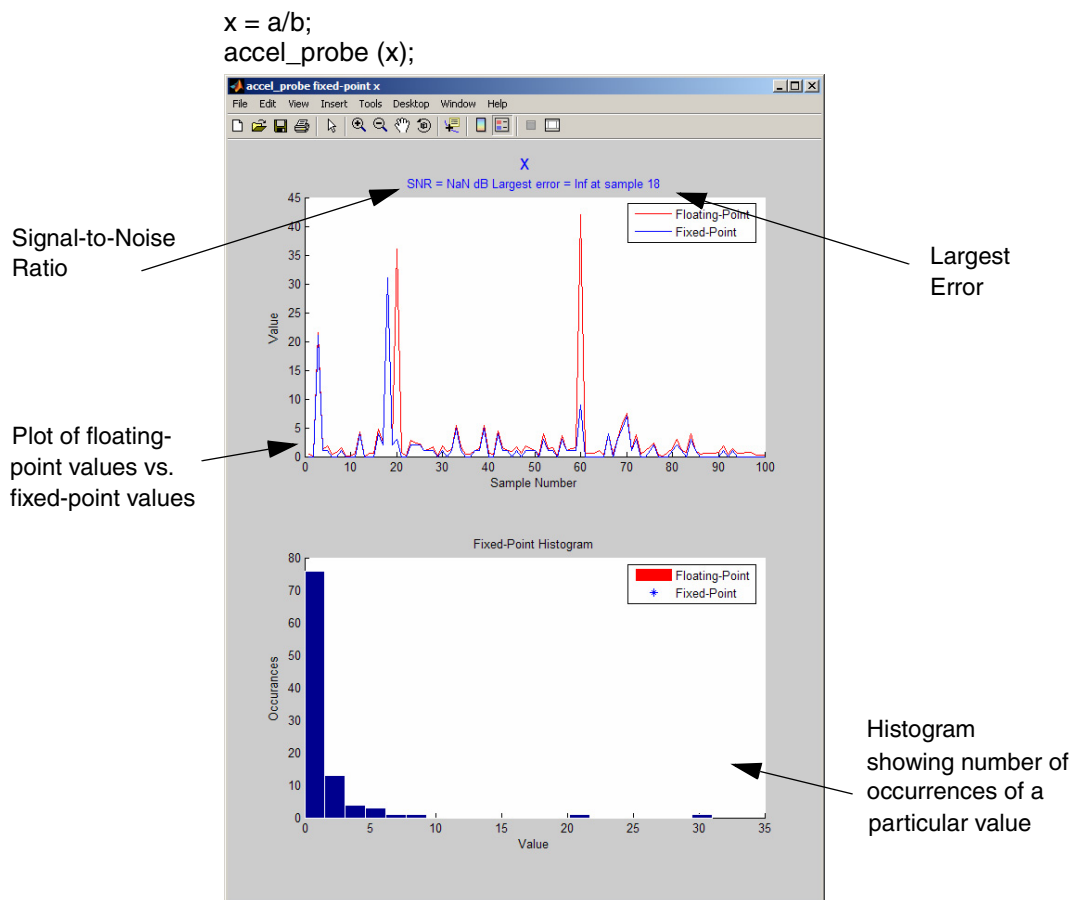


Figure 8: AccelProbe Plot for a Variable

Summary

When inventing the mathematics of a DSP algorithm, MATLAB is the natural choice and should be used unencumbered by hardware considerations. Converting that 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.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/19/06	1.0	Initial Xilinx release.