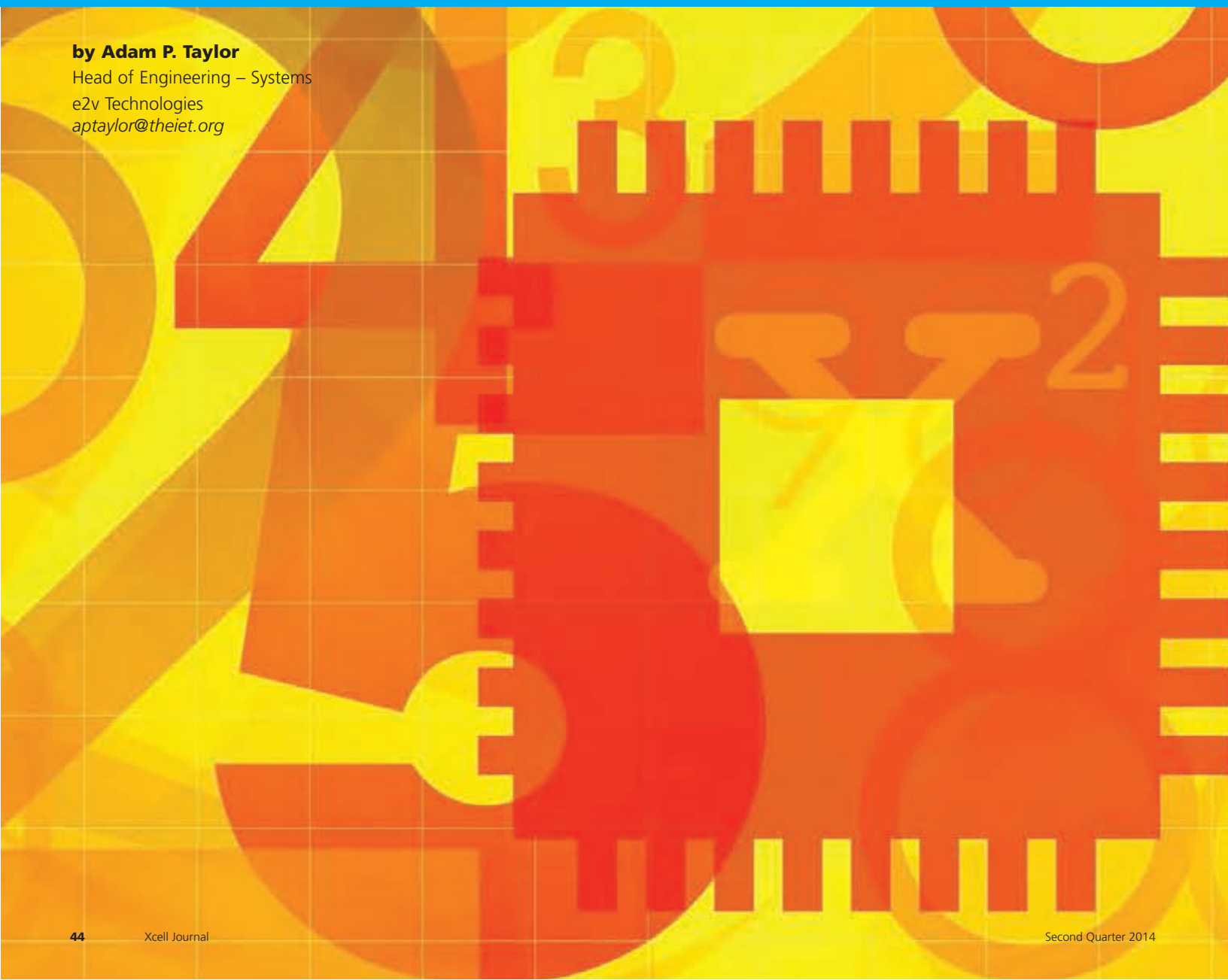
Calculating Mathematically Complex Functions

by Adam P. Taylor
Head of Engineering – Systems
e2v Technologies
aptaylor@theiet.org



One of the great benefits of FPGAs is that you can use their embedded DSP blocks to tackle the knottiest mathematical transfer functions. Polynomial approximation is one good way to do it.

Thanks to their flexibility and performance, FPGAs have found their way into a number of industrial, science, military and other applications that require the calculation of complex mathematical problems or transfer functions. It is not uncommon to see tight accuracy and calculation latency times in the more critical applications.

When using an FPGA to implement mathematical functions, engineers normally choose fixed-point mathematics (see *Xcell Journal* issue 80, "The Basics of FPGA Mathematics," *http://issuu.com/xcelljournal/docs/xcell80/44?e=2232228/2002872)*. Also, there are many algorithms, such as CORDIC, that you can use to calculate transcendental functions (see *Xcell Journal* issue 79, "How to Use the CORDIC Algorithm in Your FPGA," *http://www.xilinx.com/publications/archives/xcel l/Xcell79.pdf)*.

However, when confronting functions that are very mathematically complex, there are more efficient ways of dealing with them than by implementing the exact demanding function within the FPGA. To understand these alternative approaches—especially one of them, polynomial approximation—let us first define the problem.

LAYING OUT THE PROBLEM

One such example of a complex mathematical transfer function would be within an FPGA that monitors a platinum resistance thermometer (PRT) and converts the resistance of the PRT into a temperature. This conversion typically occurs using a Callendar-Van Dusen equation. In its simplified form, shown below, this equation can determine temperatures between 0°C and 660°C.

$$R = R_0 \times (1 + a \, x \, t + b \, x \, t^2)$$

where R_0 is the resistance at 0°C and a and b are coefficents of the PRT and t is the temperature.

In reality, we want to go from a resistance to a temperature. To do so, we need to rearrange the equation so that the result is the temperature for a given resistance. Most systems that use a PRT will design electronics to measure the resistance of the PRT using an electronic circuit, leaving the FPGA to cal-

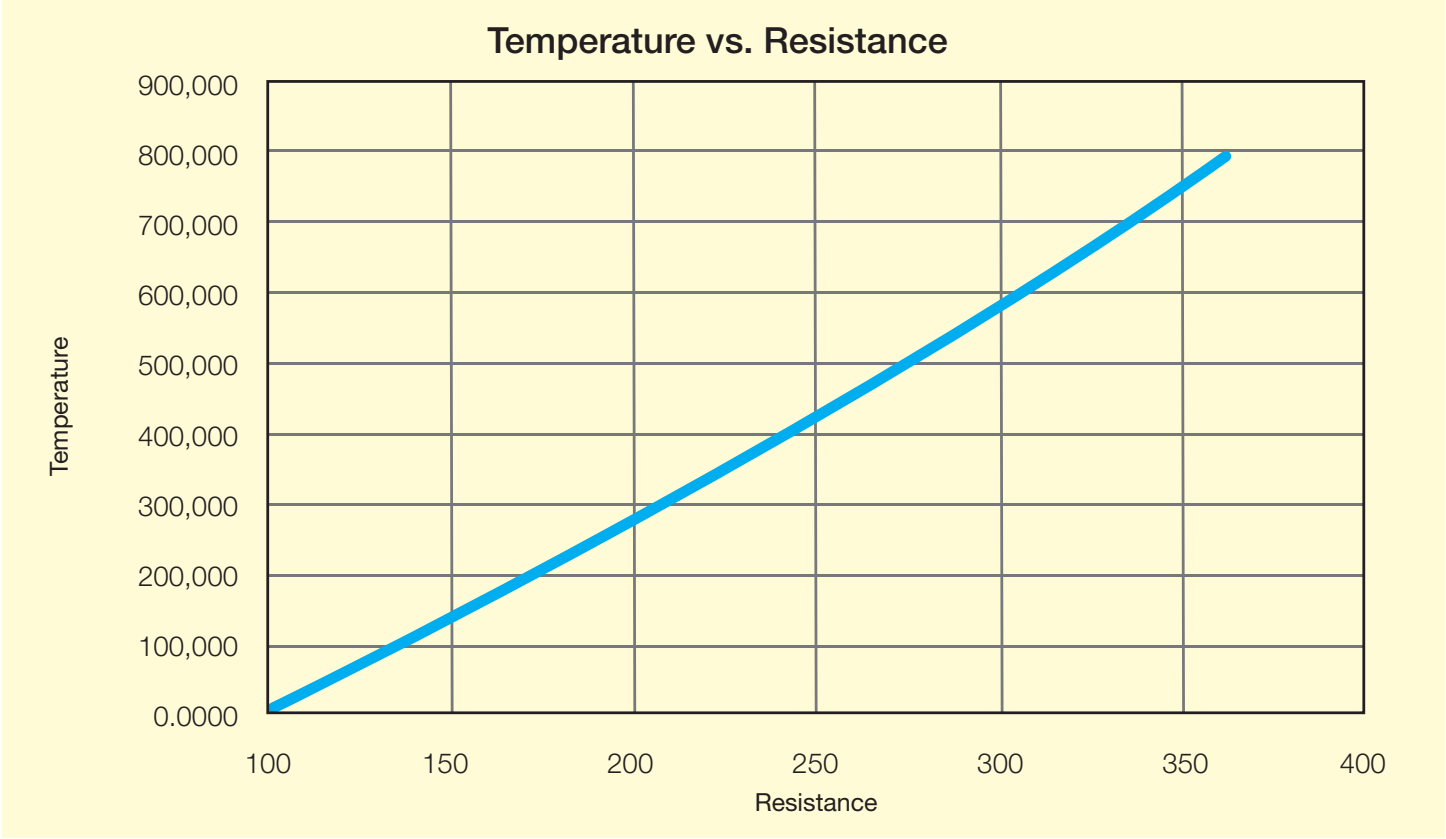


Figure 1 – The plotted transfer function

culate the temperature using the rearranged equation below.

$$t = \frac{-R_0 \times a + \sqrt{R_0^2 \times a^2 - 4 \times R_0 \times a \times (R_0 - R)}}{2 \times R_0 \times b}$$

Implementing this equation within an FPGA may be daunting for even a seasoned FPGA engineer. Plotting the obtained resistance against temperature results in a graph as shown in Figure 1. From the graph, you can clearly see the nonlinearity of the response.

Implementing the rearranged transfer function in an FPGA directly could be a significant challenge, both in terms of actual design effort required and then in validation (ensuring accuracy and function across boundary and corner-case conditions). Many engineers will look for different methods to implement the function in a way that will reduce design and validation effort so as to protect project time scales. One possible approach would be using a lookup table to store a number of points on the curve with linear interpolation between the points within the LUT.

This approach may fit the bill, depending upon the accuracy requirement and number of elements stored within the lookup table. However, you will still need to include a linear interpolator function within the design. This function can be mathematically sophisticated and will often include a non-power-of-two divide, which adds to the complexity.

CAPITALIZE ON FPGA RESOURCES

Instead, there is another method you can use to implement these types of transfer functions—one that capitalizes upon the very nature of the FPGA. Modern FPGAs like the Xilinx® Spartan®-6 and the 7 series Artix®, Kintex® and Virtex® lines contain much more than just the traditional lookup tables and flip-flops. They also come with built-in DSP slices, Block RAM and distributed RAM, along with many advanced hard IP cores such as PCIe® and Ethernet endpoints, high-speed serial links and so on.

Engineers often call the DSP slices DSP48s, due to the 48-bit accumulator they provide. However, these slices also supply 25 x 18-bit-wide multipliers and addition/subtraction capabilities, among many other faculties. It is these internal RAM structures and DSP slices that you can use to implement transfer functions with greater ease.

POLYNOMIAL APPROXIMATION

One method that utilizes the DSP- and RAM-rich architecture of the FPGA is polynomial approximation. To use this technique, you must first plot the mathematical function, covering the input value range in a mathematical program such as MATLAB® or Excel. You can then add a polynomial trend line to the data set in question, such that the equation for the trend line can then be implemented within the FPGA in place of the mathematically complex function, provided the trend-line equation meets the accuracy requirements.

Should one polynomial equation not provide sufficient accuracy over the entire transfer function input range, just add more. You can still rely on this approach so long as you generate a number of polynomial constants for use across the input range.

Most mathematical programs capable of adding a polynomial trend line allow you to select the order, or number of polynomial terms. The larger the order, the more accurate the fit should be—but the more terms you will need to implement within the FPGA. When performing this process for the transfer function example we are using in Microsoft Excel, we obtained the trend line and equation seen in Figure 2. This example used the polynomial order of four.

Having obtained the polynomial fit for the transfer function we want to implement, we can then double-check for accuracy against the original transfer function using the same analysis tool, in this case Excel. For the case in point where temperature is being monitored, it may be that the end measurement has to be accurate to +/-1°C, not a particularly demanding accuracy requirement. Still, it may prove difficult to achieve using just one polynomial equation, depending upon the range of measurements and the transfer function you are implementing. How can we address this problem?

MULTIPLE TREND LINES SELECTED BY INPUT VALUE

Should one polynomial equation not provide sufficient accuracy over the entire transfer function input range, just add more. It is still possible to

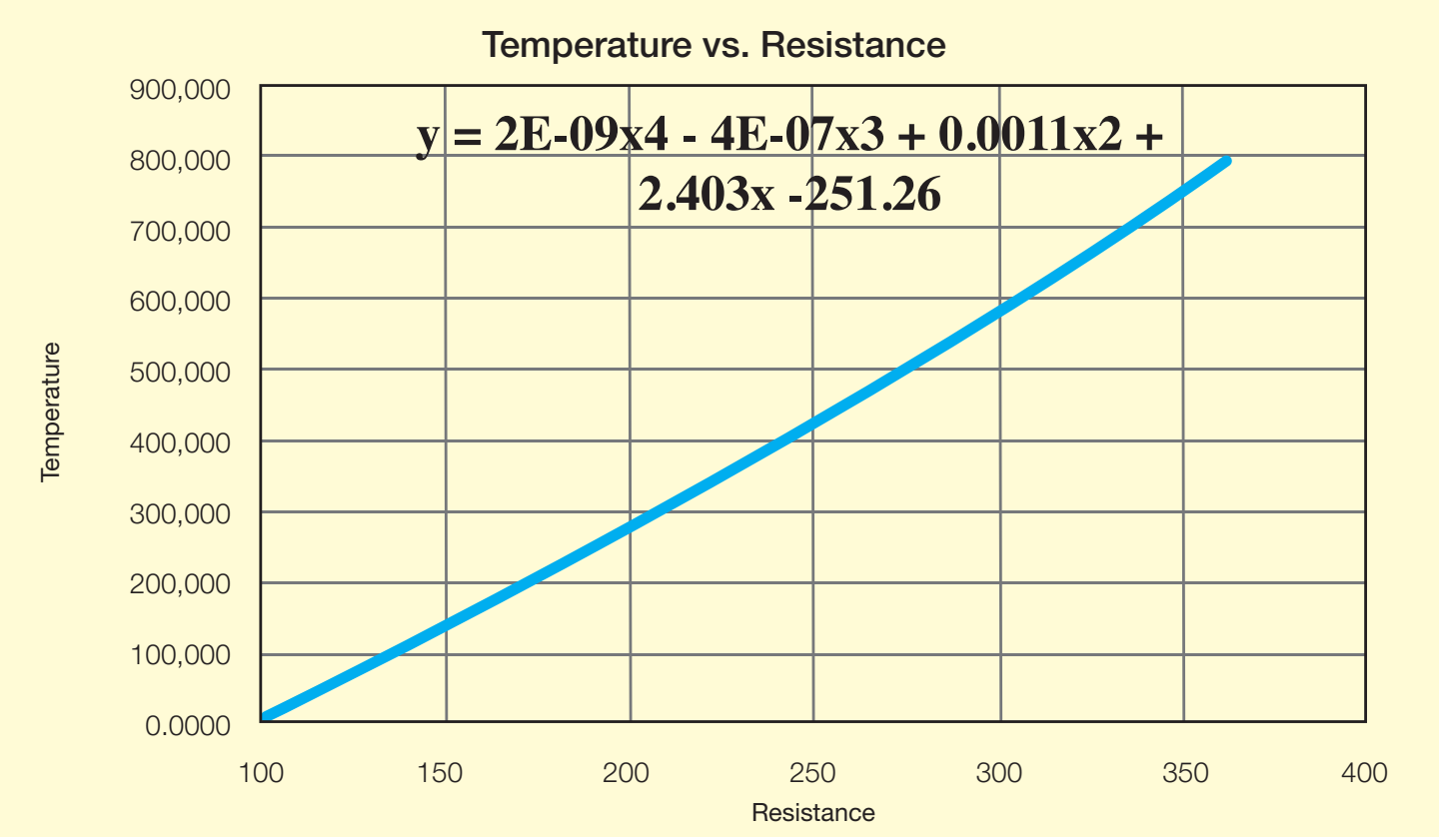


Figure 2 – Trend line and polynomial equation for the temperature transfer function

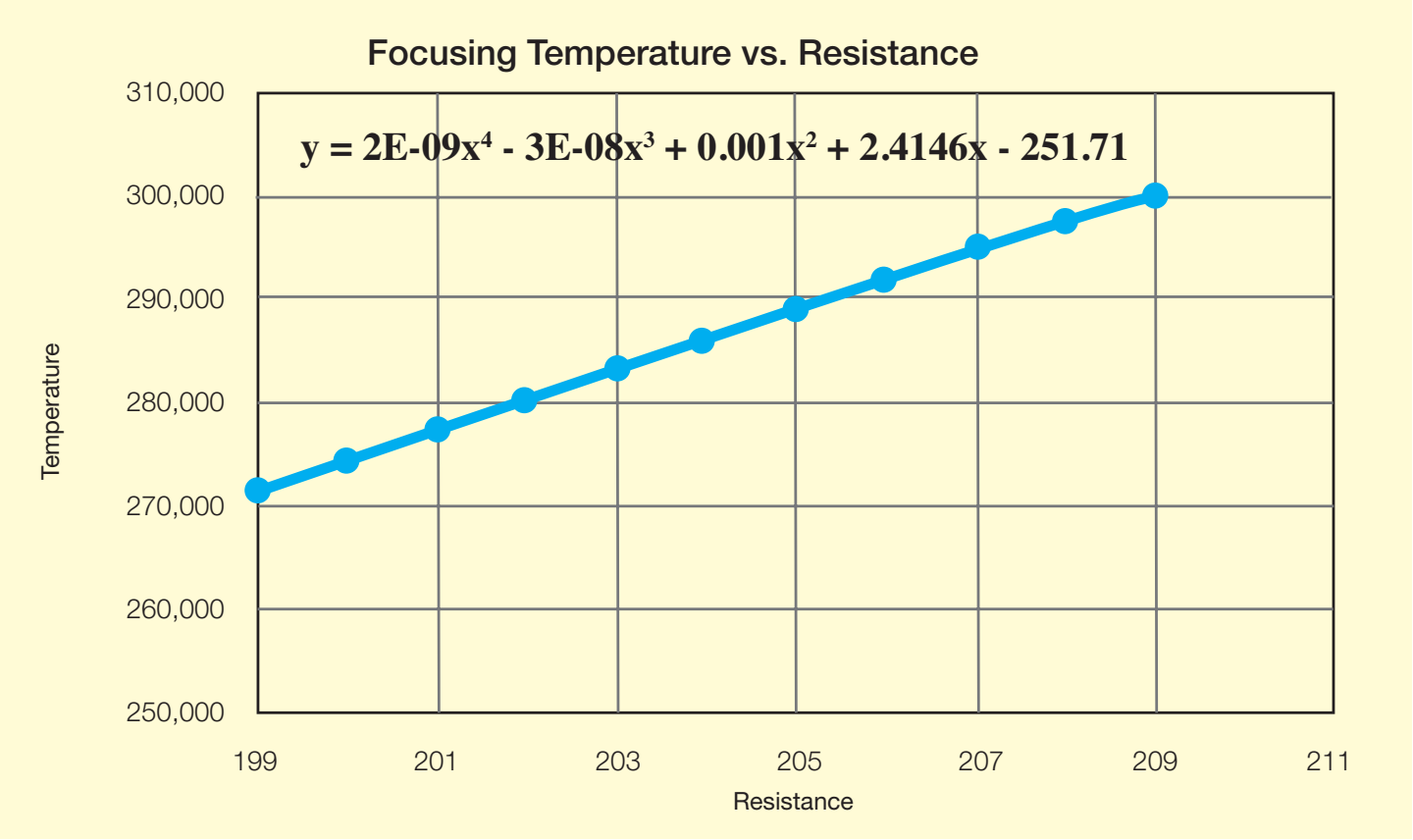


Figure 3 – Plotting between 269°C and 300°C to provide a more accurate result

use this approach so long as you generate a number of polynomial constants for use across the input range. Thus, once the input value goes outside specific bounds, a new set of constants is loaded.

Continuing with the temperature example, the first polynomial equation provides +/- 1°C of accuracy between 0° and 268°C. For many applications this will be more than sufficient. Suppose that we require an extended operating range and tolerance to 300°C, which would mean our initial approach did not meet the design requirements. Using a segmented approach, we can address this problem by plotting the range between 269°C and 300°C and obtaining a different polynomial equation that will provide more accuracy for this output range (see Figure 3).

In short, the implementation uses the first polynomial constants until the input value goes above a precalculated range that corresponds to 268°C. Above that range, the second set of constants is used to maintain the accuracy requirements.

In this way, you can break a transfer function into a number of segments to achieve the desired accuracy. You may opt to space these segments uniformly across the transfer function—that is, split them into 10 segments of equal value of X. Alternatively, you can make them nonuniform and segment them as required to achieve the desired accuracy, focused upon areas of the transfer function where accuracy is harder to obtain.

Among the trade-offs to consider when deciding upon your implementation, keep in mind that a uniform approach may require a larger memory footprint than a nonuniform approach. Depending upon the transfer function you are implementing, going with a nonuniform approach could result in a considerable saving.

HOW DOES IT COMPARE?

Of course, as I mentioned earlier there are other methods you can use to implement transfer functions. The four most commonly used methods aside from polynomial approximation are software routines, lookup tables, a lookup table with interpolation and CORDIC.

The use of software to calculate the transfer function complicates the system architecture due to the need to add a processor (with an associated increase in design complexity, BOM cost and so on). Even if the design team mitigates this drawback by using a system-on-chip such as the Xilinx Zynq®-7000 All Programmable SoC, challenges still remain. For starters, the time it takes to calculate the transfer function in software would be much longer than can be achieved in logic, reducing the system response time. In fact, calculation of transfer functions such the one used in our sample design is a classic exam-

Polynomial approximation presents a middle ground among the four alternative methods of implementing transfer functions, offering a good trade-off in performance, accuracy and footprint.

ple of where the processing should be offloaded to the programmable logic side of the Zynq SoC.

The efficacy of the second approach—using a lookup table containing precalculated values for the input—can vary depending upon the range and width of the input values. At times, the result will very quickly be a very large LUT that requires a lot of RAM within the FPGA. Depending upon the FPGA, this approach could require more resources than are available, or it might cause conflicts with requirements for other modules within the design. On the plus side, of course, this method will result in a very fast "calculation" of the result.

The third potential approach—a lookup table with interpolation—is one we explored earlier and is an attempt to reduce the number of memory locations needed with a full LUT approach. This technique does require the engineer to write a linear interpolation function within the FPGA, which can be a little involved. It is, however, still much simpler than the final option: CORDIC.

A CORDIC algorithm is capable of implementing transcendental functions such as sine, cosine, multiplication, division, square root and so on. Therefore, it is possible to implement transfer functions exactly using a combination of CORDIC algorithms and basic mathematical blocks. The technique can result in higher precision. However, for a complicated transfer function, this gain in precision will come at the cost of increased design and verification time. There will of course be an impact on the operating frequency of a device implemented in this way.

Polynomial approximation therefore presents a middle ground among the four alternative options, offering a good trade-off in performance, accuracy and implementation footprint.

EASE OF IMPLEMENTATION

Every engineer wants to produce an FPGA that has an optimal utilization of device resources. Polynomial approximation allows you to benefit from the multiplier- and RAM-rich environment provided by the FPGAs, and to use these resources to easily implement what at first might appear to be a very complex mathematical transfer function.