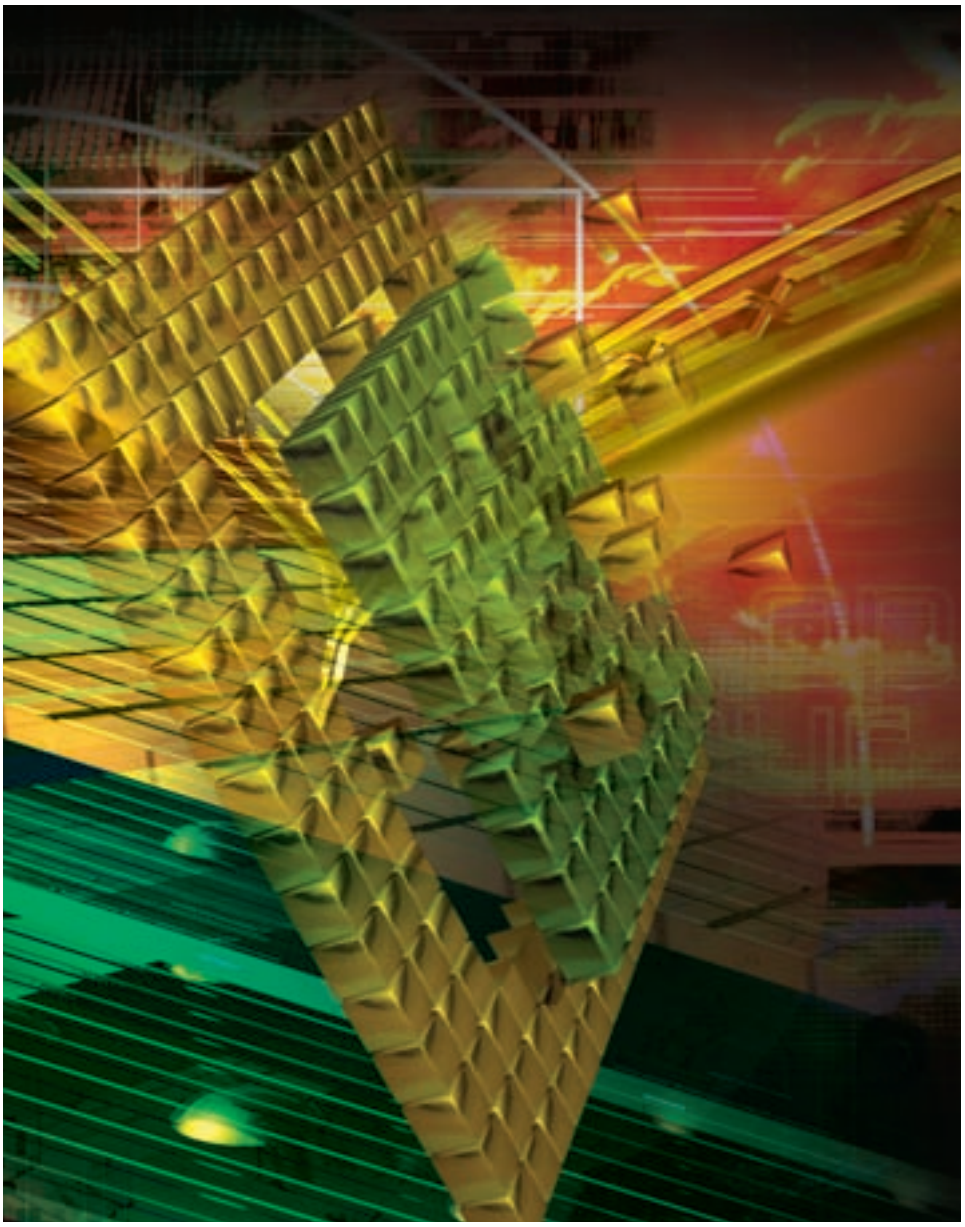
Supercharge Your DSP with Ultra-Fast Floating-Point FFTs

A hybrid technique yields a gigasample-per-second 32- to 2,048-point IEEE floating-point FFT in a Virtex-4 SX device.



by Ray Andraka
President
Andraka Consulting Group, Inc.
ray@andraka.com

Engineers targeting DSP to FPGAs have traditionally used fixed-point arithmetic, mainly because of the high cost associated with implementing floating-point arithmetic. That cost comes in the form of increased circuit complexity and often degraded maximum clock performance. Certain applications demand the dynamic range offered by floating-point hardware but require speeds and circuit sizes usually associated with fixed-point hardware. The fast Fourier transform (FFT) is one DSP building block that frequently requires floating-point dynamic range and high speed.

A textbook construction of a pipelined floating-point FFT engine capable of continuous input entails dozens of floating-point adders and multipliers. The complexity of these circuits quickly exceeds the resources available on a single FPGA. We fit the FFT design into a single FPGA without sacrificing speed or floating-point performance by using an alternative FFT algorithm and a hybrid of fixed- and floating-point hardware.

The resulting design has IEEE single-precision floating-point inputs and outputs that match the precision obtained with more conventional designs, yet is capable of as much as 1.2 gigasamples-per-second continuous data throughput and fits into one Xilinx® Virtex™-4 XC4VSX55 FPGA. The design performs 32-, 64-, 128-, 256-, 512-, 1,024-, or 2,048-point complex input Fourier transform, with size selected on the fly.

The Algorithm

The FFT can be factored in a variety of different ways; each way results in a different algorithm. The most common factorization is the Cooley-Tukey algorithm, which recursively factors each N point transform into a pair of N/2 point transforms combined by a "butterfly" operation until the reduced transforms are each a single sample long. Each butterfly operation comprises a complex multiply by a phasor or "twiddle factor" (that is, a phase rotation of the input) followed by a two-point FFT, which is just a complex sum and difference. The regularity of the algorithm and data sequencing is ideal for a software implementation. The number of multiplies and floating-point operations for a software implementation are of little consequence because all of the operations are performed serially by a single floating-point unit.

Other factorizations of the Fourier transform can result in a more efficient hardware design. The Winograd FFT algorithm is particularly interesting for hardware implementations because it is a factorization designed to minimize the number of multiplies needed to perform the transform. The price you pay for the reduced number of operations is a highly irregular addressing sequence, which makes it very inefficient to perform with a microprocessor.

A hardware solution using distributed memory mostly avoids the problems posed by an irregular address sequence. A 16-point Winograd FFT decomposes into three layers of add/subtract operations on each side of a real multiplier (Figure 1), which performs 74 add and 18 multiply operations per FFT. This represents about one third of the hardware of a Cooley-Tukey implementation, which requires 176 add and 72 multiply operations. Andraka Consulting Group has developed optimized fixed-point Winograd kernels for maximum performance 4-, 8-, and 16-point FFTs targeted at all Virtex families. This particular design uses a redesign of our Winograd kernel that takes advantage of the DSP48 primitives; is capable of selectable 1-, 4-, 8-, and 16-point operation; and can operate at clock rates as high as the maximum clock rate of the DSP48 slice (400 MHz in the -10 speed grade device).

The Winograd FFT is awkward to factor for sizes larger than 16 point, so rather than attempting to compute the full FFT directly, we use the mixed-radix algorithm to construct larger FFTs from small 4-, 8-, and 16-point Winograd FFT kernels. The mixed-radix algorithm arranges the data into a matrix, performing a small FFT down each column, phase-rotating the results, and then doing an additional FFT across each row. The 32- to 2,048-point design uses the mixed-radix algorithm to obtain a 32-, 64-, 128-, or 256-point FFT from a pair of 4/8/16 kernels, and then uses the algorithm a second time to combine that composite FFT with an additional 8-point kernel to obtain the 512-, 1,024-, and 2,048-point FFTs (Figure 2). This design is easily extended to 4,096 points by using a 16-point kernel for the extra stage and may be extended to larger FFTs using additional stages, rotators, and reorder memory.

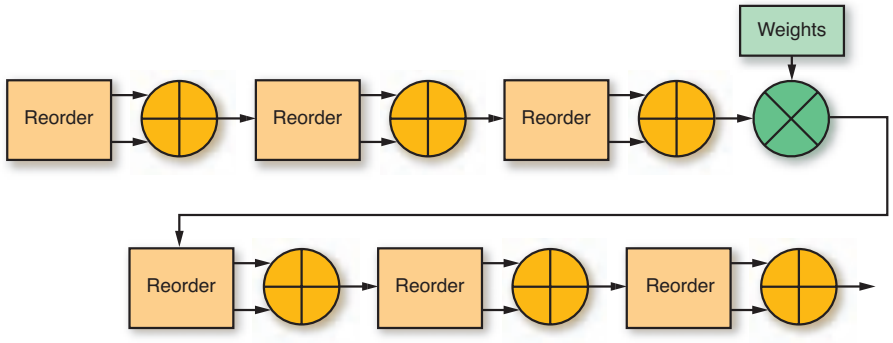


Figure 1 – A 16-point Winograd FFT

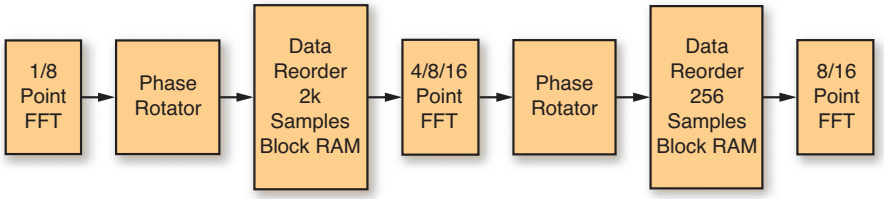


Figure 2 – Combination of small Winograd kernels using mixed-radix algorithm to achieve larger FFTs

Hybrid Fixed Point/Floating Point

Traditional floating-point implementations treat each add or multiply operation as a stand-alone floating-point operation requiring normalized inputs and outputs. When these basic operations are assembled into more complicated operators, the intermediate normalize and de-normalize operations are often unnecessary and represent a considerable amount of wasted hardware.

With a hybrid approach, you can take larger pieces of the algorithm and treat them as fixed-point blocks that operate on the mantissas of the input data, renormalizing after several algorithm steps rather than

after each elemental operation. The input to the fixed-point operator has to be de-normalized so that it shares a common exponent, and the fixed-point operator must have enough extra bits to allow for any combination of inputs without overflow.

The FFT design uses Winograd FFT blocks to compute 4-, 8-, or 16-point FFTs. Considering the properties of the FFT, these blocks have a maximum gain of 16, which is accommodated by allowing for a 4-bit growth. These FFT blocks make an ideal candidate for a fixed-point function block in our hybrid floating-point approach. The design shown in Figure 3 de-normalizes each of the complex inputs to the small FFT so that the largest sample is left-justified in the 30 bits presented to the FFT.

The remaining samples are right-shifted by the difference between their exponent and the exponent of that largest sample. A new maximum is obtained for each 4-, 8-, or 16-point set so that the small FFT is always performed at the maximum possible precision. The fixed-point FFT kernel internally expands to 35 bits and presents 35 bits at the output so that overflow is not possible – regardless of the inputs. Each complex pair out of the FFT kernel is then re-normalized, producing a complex pair with a common exponent for the I and Q components. The larger component is left-justified.

Because the FFT algorithm involves addition, no precision is lost using this method when compared to performing the entire FFT with floating-point arithmetic. This is because the smaller intermediate results wind up getting rounded in a full floating-point implementation anyway, so the precision of any FFT output is no greater than the precision of the largest input.

The phase rotations are accomplished with fixed-point hardware (a 35 x 35-bit complex multiply). But because these are multiplications, they are very similar to floating-point multiplies. The input from the FFT kernel is an IQ pair with a common exponent and with the larger component left-justified. The twiddle factor is a 35-bit fixed-point sine and cosine pair. Because the selected number representation has a common exponent for the IQ pair and the twiddle factor has a unity magnitude, the result of the operation has the same common exponent. The worst-case rotations grow or shrink the larger component by one bit, requiring at most a one-bit shift to re-normalize after the rotation. The inputs and outputs from this floating-point 4-/8-/16-point kernel (Figure 3) are floating-point IQ pairs rounded to 30 bits with a common 8-bit exponent.

Using this internal pair format rather than separate exponents reduces the storage for intermediate results, makes normalization easier, and loses nothing compared to independent I and Q components. The internal representation is converted from and to IEEE single-precision format at the input and output of the FPGA, respectively.

Conclusion

We implemented this design in a Xilinx Virtex-4 XC4VSX55 -10 device. All of the arithmetic is confined to the DSP48 slices so that the design is not delayed by the relatively slow carry chains in the FPGA fabric. Careful implementation has produced a design whose maximum clock rate is the DSP48's maximum clock rate – 400 MHz.

The 32- to 2,048-point FFT described here operates at speeds as fast as 400 complex megasamples per second using a 400-MHz clock, and occupies less than 30% of the FPGA. Three instances, along with a round-robin controller, QDR memory interfaces (for the data source and sink), and buffering fit within the device. By sequencing the FFT starts, the three threads achieve a composite 1.2 gigasample-per-second continuous throughput.

Using the standard "5 N log N" metric for computing FFT floating-point operations, the one-dimensional 2,048 point FFT is computed at 22 gigaFLOPS per second in each of the three parallel engines, or a composite 66 gigaFLOPS per second. By comparison, in the article "A Programming Example: Large FFT on the Cell Broadband Engine," IBM claimed 46.8 gigaFLOPS for a single-precision FFT on the Cell Broadband Engine. The design will port easily to Virtex-5 devices, which will exhibit even greater performance and density.

For more information, please visit *www.andraka.com.*

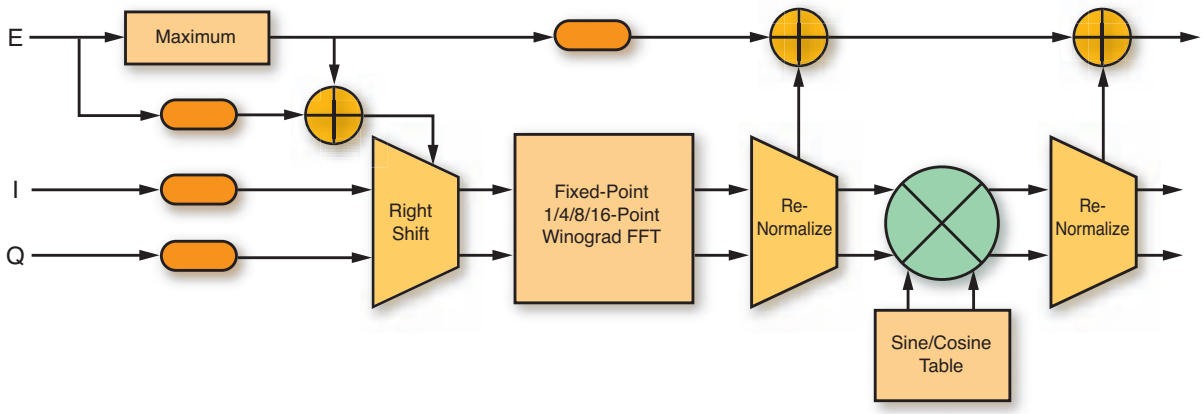


Figure 3 – Hybrid floating-point 4-/8-/16-point FFT kernel with rotator for mixed-radix cascade