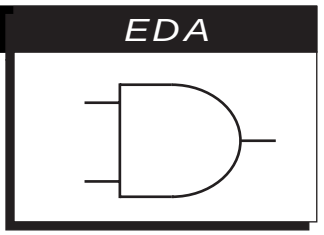

FPGAs furnish fast, furious FIR filters

Steve Knapp

A previous column (Ref 1) described a fast, efficient method to implement multipliers in a lookup-table (LUT) based field-programmable gate array (FGPA) when one of the inputs is a constant value. Building on that method, another column (Ref 2) described a method to efficiently combine numerous multiply / accumulate operations using serial distributed arithmetic. This column expands on both ideas to demonstrate how parallel processing in FPGAs boosts the performance of digital filters to new heights. It uses a finite-impulse response filter for examples, but the concepts apply to other types of digital filters.

FIR filters are a mainstay in many DSP applications, usually providing signal conditioning, antialiasing and convolution. Fig 1 shows an example 8-bit, 8-tap FIR filter. It receives a data sample X_0 on every data clock cycle. All the other values, $X_{1\text{-}7}$, are time-delayed values of X_0. The filter multiplies each data sample X_n by its corresponding constant filter coefficient H_n. It sums the products from all the multipliers to produce the result Y.

Steven K Knapp is the founder and president of OptiMagic Inc (Aptos, CA, www.optimagic.com), a firm that develops intellectual property and design software for programmable logic. Prior to founding this firm, he held various applications, engineering and management positions at Xilinx and Intel's former programmable-logic division.

In mathematical terms, you can express the filter function as the series

$$Y = \sum_{n=1}^{8} X_n H_n .$$

When implemented on a traditional DSP, the chip multiplies each sample by its corresponding filter coefficient using a single-cycle multiply / accumulate instruction. Consequently, it requires at least eight clock cycles to produce the final result—when each multiply / accumulate operation executes sequentially. Using a 66-MHz DSP with its 15-nsec cycle time produces a theoretical maximum 8.33 MHz data-sample rate (15 nsec / clock x 8 clocks / data sample = 120 nsec / data sample). This rate assumes that all values reside within the processor's internal registers. The actual data rate can become slower if the processor needs to fetch values or place results in external memory.

Compare this performance against an FPGA implementation where all multiply / accumulate operations take place in parallel (Fig 1). Each filter

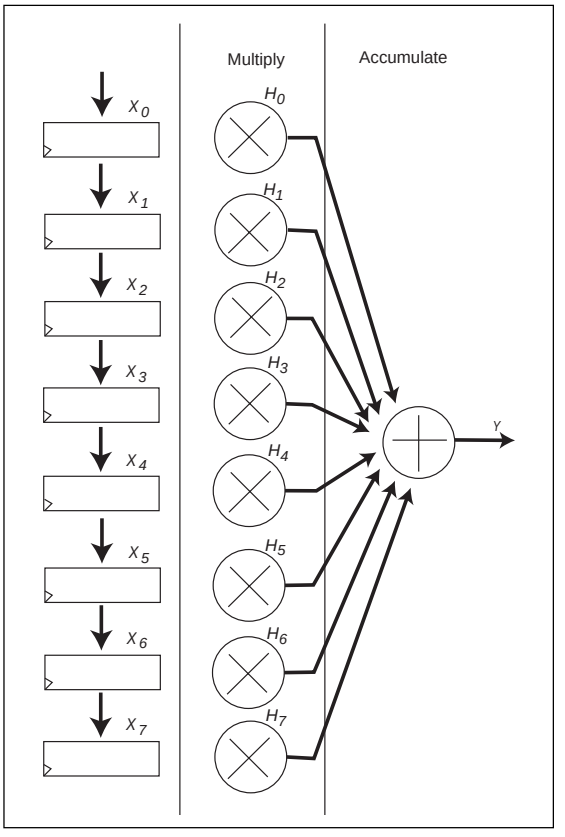

Fig 1—An example 8-bit 8-tap FIR filter receives a new data sample every clock cycle, and the other values are time-delayed values of that sample. A traditional DSP requires eight clock cycles to produce the final result.

tap multiplies the register contents by a constant. The constant-coefficient multiplier uses the method described in Ref 1. The FPGA implementation can sustain a significantly higher clock frequency, well above 66 MHz for most FPGAs, assuming a pipelined design. In this simple example, the FPGA approach outperforms a traditional DSP by a factor of eight or better.

A parallel implementation solves most performance-critical applications. However, not all designs require such high data rates. The fully parallel approach burns a significant amount of FPGA resources, especially for filters with more taps or with a wider data word. Is there a more space-efficient approach for lower data rates?

Shifting to serial

Using the serial distributed arithmetic (SDA) method described in Ref 2, you can reimplement the design in Fig 1 with a serial scheme (Fig 2). It captures the incoming data sample X_0 in a parallel-to-serial shift register. Registers $X_{1\text{-}7}$ serve as serial-to-serial shift registers. The serial output from X_0 feeds into X_1 and cascades through all the data registers until it falls out at the bottom. One bit from each of the registers $X_{0\text{-}3}$ feeds a LUT-based multiplier containing sums of the constants $H_{0\text{-}3}$. By serially shifting data through the filter, this method reduces the eight LUT-based parallel multipliers down to two SDA LUT-based multipliers.

This implementation requires 64 flip-flops (eight 8-bit registers) to capture and shift each data sample. In some FPGA architectures, such as the Xilinx XC4000, Spartan, and Virtex as well as Lucent's Orca devices, the lookup table in each logic cell can implement RAM as well as more traditional logic functions. Using these devices, designers can replace the serial-to-serial shift registers with RAM-based shift registers, again reducing resource requirements (Ref 3). Here a 16-bit shift register consumes the resources of one flip-flop.

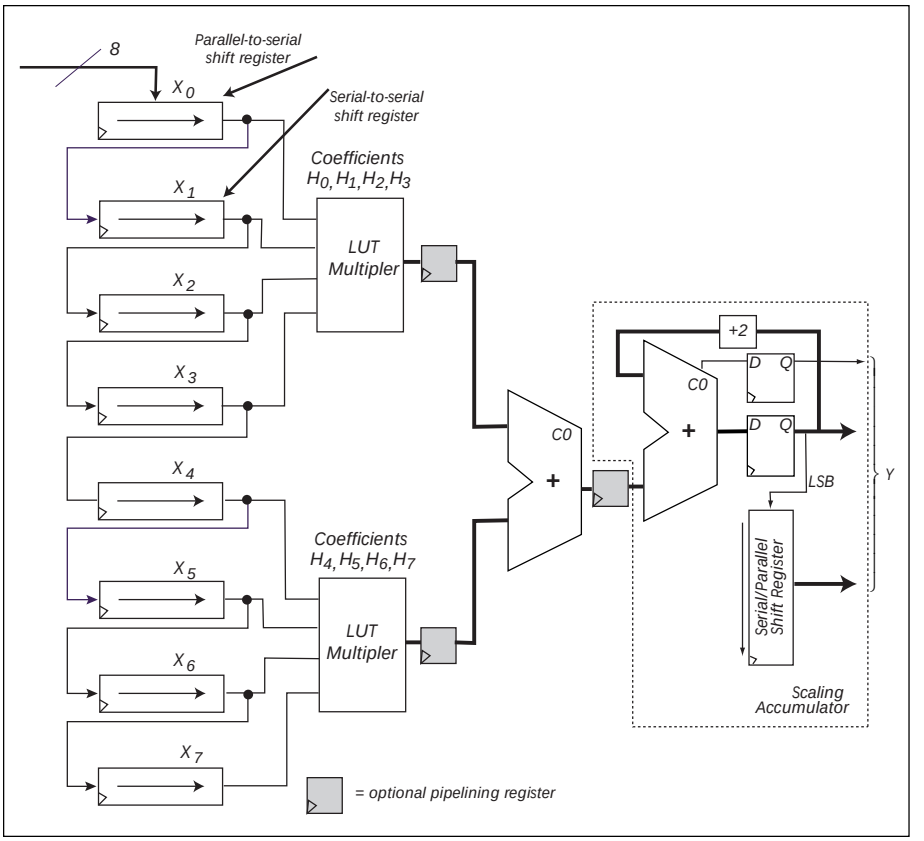

Fig 2—With the serial-distributed arithmetic (SDA) approach, the filter in Fig 1 becomes faster because this scheme determines the result in a scaling accumulator. This alternative cuts the number of constant-coefficient multipliers in half.

Exploiting symmetry

In certain cases it's possible to do even better. For instance, note that coefficients in linear-phase FIR filters are symmetrical around the center—coefficient H_0 equals H_7, H_1 equals H_6 and so on. Consequently, you can modify the equation presented earlier accordingly:

$$Y = H_0(X_0 + X_7) + H_1(X_1 + X_6) + H_2(X_2 + X_5) + H_3(X_3 + X_4)$$

This symmetry allows the scheme to add symmetric taps before it multiplies them by their corresponding coefficients, cutting the number of constant-coefficient multipliers in half. Data snakes its way through the shift registers as before. However, the outputs from two of the filter taps feeds into a serial adder. The sum, held in a flip-flop, feeds the LUT-based multiplier. The output from the multiplier feeds a scaling accumulator, as in Fig 2. An extra step flushes the final carry from the serial adder, slowing processing time by an additional clock cycle.

Making the choice

With significant performance and cost advantages for some applications, will FPGAs end up replacing DSPs? Not likely. DSPs will still make up the bulk of signal-processing applications. FPGAs, however, excel at performance-critical filtering, supporting data rates significantly higher than any existing DSP or even multiple processors.

The performance of DSPs drops with each sequential instruction. In the 8-tap filter example, a DSP could theoretically process the filter algorithm in eight clock cycles. A DSP's sustainable data rate is directly proportional to the number of filter taps as long as the device supports the data-word width (in other words, operating on a 22-bit word in a 24-bit processor):

Fig 4—In evaluating the performance of various filter implementations, it becomes clear that the FPGA becomes more attractive than a DSP with an increasing number of taps or faster processing requirements.

ART C

DSP datarate ∝ number of taps

Because an FPGA processes all taps in parallel, its performance doesn't depend on the number of taps but instead on word width. In a fully parallel implementation, the wider adder tree and scaling accumulator degrades performance. With the SDA approach, a wider word requires additional clock cycles:

FPGA data rate (SDA, nonsymmetrical) ∝ word width

FPGA data rate (SDA, symmetrical) ∝ word width + 1

Fig 4 (Ref 4) compares the performance of a DSP and an FPGA implementation. The horizontal axis represents the number of filter taps while the vertical axis gives the data sample rate, not the clock frequency. The shaded region represents the performance domain of a single 66-MHz fixed-point DSP. Additional curved lines represent the performance for 2- and 4-DSP systems. Adding multiple processors operating in parallel increases performance on filtering algorithms. However, the multiple processors and their associated high-speed memories add to system cost.

The non-shaded region represents the FPGA-solution domain where those devices' parallelism outperforms traditional DSPs. The horizontal lines indicate the performance of an 8- and

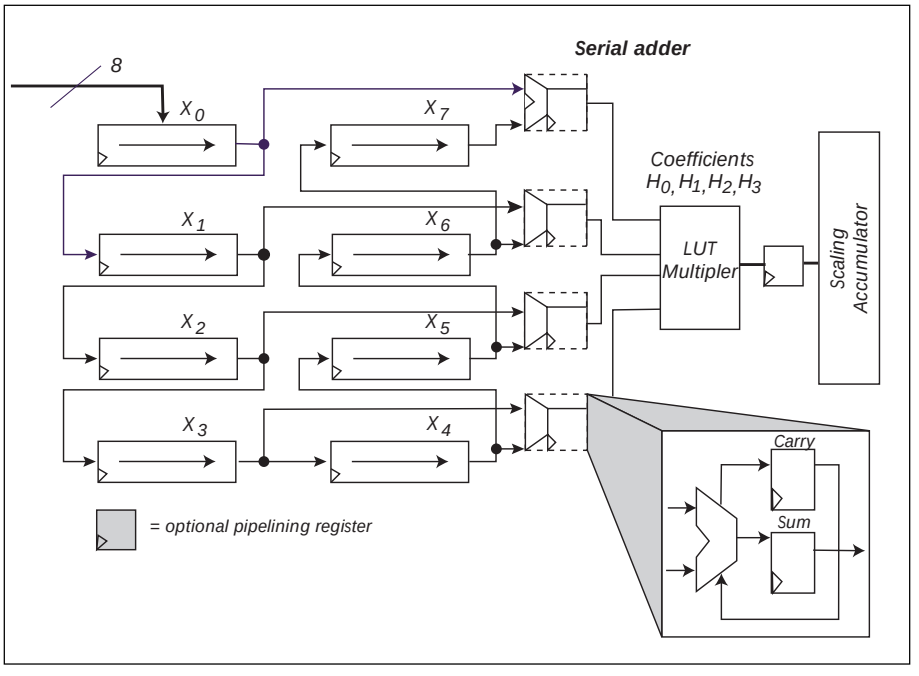

Fig 3—By turning the design of Fig 2 into a linear-phase filter, which has symmetrical coefficients, the implementation becomes somewhat simpler and requires one extra clock cycle.

16-bit FIR filter using the SDA method. Remember that the FPGA implementation slows with increasing word width, not the number of filter taps. The two top-most horizontal lines represent the SDA method but processing two and four bits in parallel. A fully parallel implementation could operate at the full 66 MHz but at the expense of additional silicon. However, a FIR filter supporting a 66-MHz sample rate might fit into one FPGA but the same filter implemented using DSPs requires a separate 66-MHz chip operating on each filter tap—plus its associated memory.

As an example, assume that an application requires a 32-tap filter for 8-bit data while supporting a 10-MHz sample rate (see large dot in Fig 4). This point falls outside the performance range for a 66-MHz traditional DSP and even for four DSPs operating in parallel. It's also too fast to work with the standard SDA approach. However, by operating on two bits in parallel—the line labeled 2 x 8-bit DA—you can hit the required performance level.

Increased performance does carry a price. The higher-performance FPGA algorithms use additional resources. Fig 5 shows the relative performance and resources required of different 8-bit FPGA-based symmetrical FIR filters. The smallest and lowest performance implementation is the SDA approach, supporting a 7.3-MHz data rate from a 66-MHz system clock. The next step up in performance processes two bits at a time and supports a 14.7-MHz data rate. The fully parallel implementation, processing all eight bits in parallel, supports a 66-MHz data rate but with increased resource requirements. The resources in this example are Xilinx-style CLBs (configurable logic blocks). Horizontal lines (labeled XCS05, XCS10 and XCS20) indicate the size of the three smallest Xilinx Spartan devices.

For the traditional DSP user, the task of designing an FPGA-based implementation might seem daunting. Luckily, the vendors listed in Table 1 either supply design software to assist in building filters or provide application notes. ***PE&IN***

Vendor	Product Family	Tool	Additional Information
Xilinx	XC4000 Spartan Virtex	CORE generator	www.xilinx.com/products/logicore/coregen
Altera	FLEX 10K FLEX 8000	DSP megafunction development kit	www.altera.com/html/mega/mega_devkit.html
Lucent	ORCA	DSP finite impulse Response filter kit	www.lucent.com/micro/fpga/dsp.html
Atmel	AT40K	Application note	www.atmel.com/atmel/acrobat/dsp40k.pdf

Table 1—FPGA-based FIR filter design resources available from device vendors

ART D

Fig 5—The resources required for various FPGA-based 8-bit symmetrical filters vary widely by device architecture.

References

1. Knapp, S, "Constant-coefficient multipliers save FPGA space, time," *PE&IN*, July 1998, pgs 45-48 (*www.pein.com/1998/PEIN0798/0798ecs.pdf*).
2. Knapp, S, "Parallel processing in FPGAs rivals DSP speed," *PE&IN*, Oct 1998, pgs 60-65 (*www.pein.com/1998/PEIN1098/1198ecs.pdf*).
3. Alfke, P, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators," Xilinx Inc, XAPP 052, July 1996 (*www.xilinx.com/xapp/xapp052.pdf*).
4. Goslin, G R, "A Guide to Using FPGAs for Application-Specific Digital Signal Processing," Xilinx Inc (*www.xilinx.com/appnotes/dspguide.pdf*).