# Use FPGA resources to boost radar system performance

This article discusses critical design trade-offs and obstacles that must be overcome to ensure a successful FPGA implementation for radar pulse compression. In addition, it discloses the development of a general-purposed intellectual property (IP) FPGA core for pulse compression.

By Rodger H. Hosking

Modern radar systems engineers seek to dramatically improve performance, target acquisition, tracking and identification of their targets. The latest generation of field-programmable gate arrays (FPGAs), with powerful new features, have become the fundamental building blocks in advanced radar platforms. FPGAs are enhancing radar system performance levels through optimized intellectual property (IP) core implementations for critical compute-intensive digital signal processing algorithms such as pulse compression and fast Fourier transform (FFTs).

Providing both increased performance and fast interface connections, FPGAs are vital in the fundamental equation of a successful radar development platform. Diverse radar system design considerations such as dynamic range, receiver noise reduction, co-site interference, signal processing, accuracy and multitarget detection can all be enhanced with the additional capabilities FPGAs can offer.

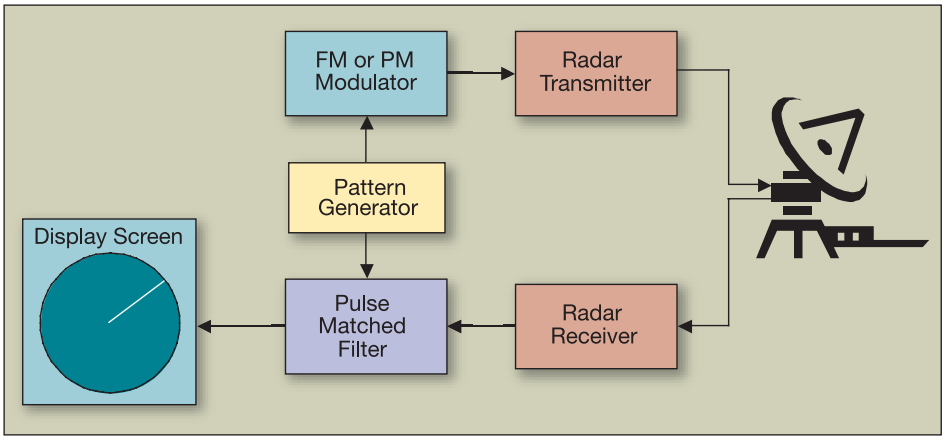


**Figure 1. Pulse compression radar.**

## Radar pulse compression basics

Early radar systems transmitted a strong pulse of RF energy and displayed reflections of the pulse on the familiar circular display screen, whose scanning beam matched the angle of the rotating dish antenna. The phosphor "blip" on the radar screen appeared at a radial distance from the screen center directly proportional to delay time of the reflected signal, and hence its distance. Range and resolution of these fixed-frequency pulse systems were limited by their peak power levels and pulse widths, respectively. Resolution could be improved by narrowing the pulse, but this reduced the outgoing peak energy resulting in compromised range performance, and also required wider bandwidth operation for both the transmitter and receiver systems.

Pulse compression is a technique that helps overcome these limitations. Instead of a fixed frequency pulse, the transmitted pulse is modulated by a specific phase or frequency pattern during a wider pulse interval. The receiver uses a pulse-matched filter to pass reflected pulses that match the pattern of the outgoing pulse and reject noise and other signals. Since the transmitted pulse is wider, a lower peak power output stage can deliver the same amount of transmitted pulse energy to maintain range performance. Figure 1 shows a basic block diagram of the system.

One popular form of pulse compression modulation is the linear frequency sweep, or chirp. The pulse-matched filter in the receiver implements a form of correlation to produce a narrow output pulse only when the received signal contains the exact frequency chirp pattern in the transmit pulse. In this way, the wide transmitted pulse is effectively compressed to a narrow pulse at the output of the correlator. The ratio of the transmitted pulse to the compressed pulse, known as the pulse compression ratio, is equal to B·T, where B is the bandwidth of the sweep and T is the transmitted pulse width.

With the narrower compressed pulse, resolution is improved dramatically and reasonable range performance can be achieved with low-power transmitters. This allows dramatic improvements for all radar systems, especially for airborne applications where size, weight and power are critical factors.

This vital advantage obviously mandates increased complexity in the signal processing stages of the transmitter and receiver. Hence, radar has been one of the prime motivators for advancements in digital signal-processing technology.

## Implementing a pulse-matched filter

One popular method of implementing a pulse-matched filter takes advantage of a well-known DSP technique: correlation of time domain signals can be achieved by a multiplication in the frequency domain. Intuitively, the frequency domain representation of two correlated time waveform signals will have identical frequency domain signatures for the portion of the signals with matching patterns. By multiplying the two frequency domain vectors (with a complex conjugate applied to one of the vectors), the resulting product will produce a match that is independent of the time alignment between the two signals. So regardless of when the reflected radar signals are received, the pulse matched filter will respond uniquely for each target returning energy. When this product vector is converted back to the time domain, each target will produce a narrow pulse whose delay and amplitude correspond to target distance and size, respectively.

Since the FFT converts time domain signals to frequency domain signals, and the inverse FFT (IFFT) performs the opposite conversion,

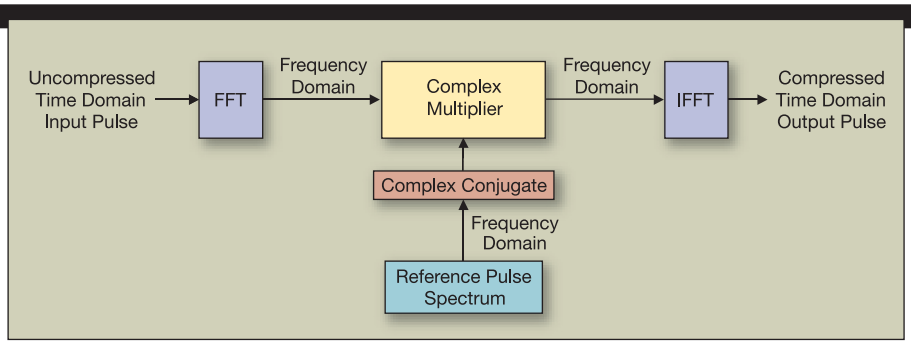


**Figure 2. Basic pulse compression core.**

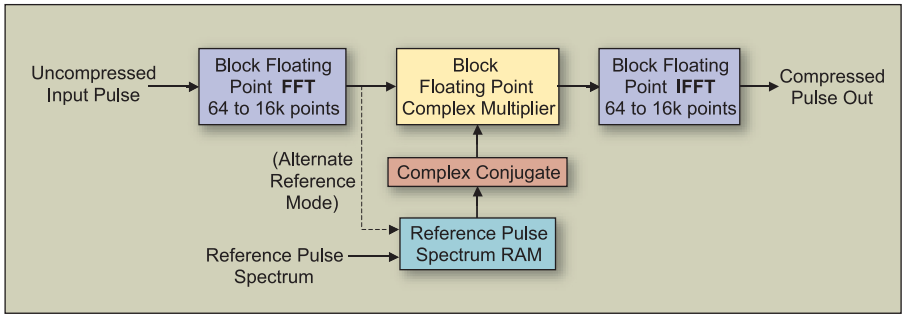


**Figure 3. Maximum performance pulse compression Core 440.**

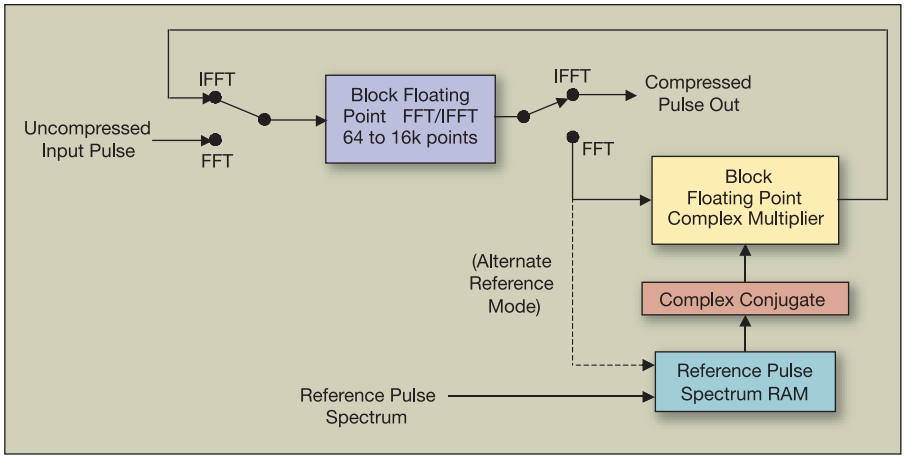


**Figure 4. Minimum resource pulse compression Core 440.**

these two algorithms are key blocks in the pulse compression system.

Figure 2 shows a complete digital pulse compression block with the FFT at the input to process radar receiver signals. In the center, the frequency domain image of the transmitter modulation pattern is stored as the reference pulse spectrum. Its complex conjugate is multiplied by the frequency domain signal from the FFT to accomplish the correlation function. The IFFT stage at the right produces the final time domain pulse compressed output signal.

## FPGA-based pulse compression radar

The FFT algorithm is usually the most critical operation in pulse compression and, therefore, FFT benchmarks are consistently used to compare performance among DSP chips.

Since FFTs involve a tremendous number of multiplications, the appearance of dedicated hardware multipliers was the single most important factor in DSPs to set them apart from conventional microprocessors. With the advent of dedicated hardware multipliers in FPGAs, these devices were soon challenging general-purpose programmable DSPs for signal processing tasks in many DSP applications, especially radar.

Instead of the one to four multiplier engines found in most DSPs, FPGAs are now sporting dozens and even hundreds of dedicated hardware multipliers. Compared with the iterative multiplications performed by program loops in DSPs, multiplications in FPGAs can be executed in parallel to deliver unprecedented FFT benchmarks.

Nevertheless, critical design trade-offs and obstacles must be evaluated and overcome to ensure a successful FPGA implementation for radar pulse compression. This article discusses these issues as they were identified and addressed during the development cycle of a general-purposed IP FPGA core for pulse compression.

## Design trade-offs and issues

Two fundamental properties of any DSP algorithm are speed and accuracy. A third factor for FPGA designs is the number of resources (gates, slices, multipliers, etc.) consumed. Unlike DSPs with fixed hardware resources, FPGAs are offered as a family of devices, whose members contain hardware resources ranging in quantity by more than an order of magnitude. Since algorithm speed and accuracy can be traded off for the number of hardware resources, many different architectural choices may be required to maximize performance for the size, cost and power constraints of specific FPGA family members. For this reason, a general-purpose IP core should be scalable for size vs. performance.

■ **Operating modes.** In order to handle diverse classes of targets and a variety of mission objectives, pulse compression systems often need multiple operational modes to support FFTs of different lengths, a range of pulse repetition rates, and several levels of dynamic range. While FPGAs can be reprogrammed by downloading a new configuration code for each mode, it's better to include support for all required modes within a given FPGA design. In this way, the operator can switch quickly and easily between modes by passing control parameters to FPGA registers.

■ **Dynamic range.** The frequency chirp is one of the most commonly used modulation patterns for radar, and its energy is inherently spread across a frequency band. Since the first stage of pulse compression is an FFT, this chirp signal results in energy distributed over many FFT bins, with relatively low energy levels in any single bin. At the same time, radar systems must be able to accommodate strong, fixed frequency signals from extraneous interfering sources without overloading.

This imposes tough dynamic range requirements at the output of the FFT stage. With enough headroom to handle the strong signals, the smaller distributed frequency components of the chirp still need enough bits of resolution to provide accurate correlation in the following stages.

True floating point processing solves this dynamic range problem quite well. However, FPGAs are quite inefficient in implementing floating point operations, since the native hardware is fixed point. In the Xilinx Virtex-II family, for example, the dedicated hardware multipliers in FPGAs are fixed point engines, accepting two18-bit inputs and producing a 36-bit product. Although a 36-bit result at the FFT output may have enough dynamic range, with this type of multiplier the subsequent conjugate multiplication stage (see Figure 2)

can accept only the 18 most significant of these 36 bits. This can result in loss of critical signal energy of the smaller chirp signal components, especially for wide chirp bandwidths.

Combining three or more 18 x 18 multipliers with some additional logic can create higher-precision fixed-point multipliers, but this quickly consumes multipliers and also adds pipeline delays to slow speed performance.

■ **Speed.** Pulse compression radar systems must be capable of processing all reflections from an outgoing pulse in a given stage before signals arrive from the next pulse. Certain modes of operation require a fast pulse repetition rate, which drives the processing speed requirements of the pulse compression engine.

FPGAs operate as synchronous state machines using a system clock to propagate data into registers between logic stages. A significant portion of the FPGA design effort involves minimizing digital signal paths so that the system clock can be increased to reduce processing time.

Factors that affect propagation delays are logic complexity (several levels of gate logic), extended precision arithmetic (as discussed above with the higher precision multipliers), and the basic speed of the silicon. Complex logic and extended precision arithmetic blocks can sometimes be partitioned into multiple clocked stages in order to boost the clock speed, but this added latency might impact a critical speed path. All FPGA vendors offer devices in a range of silicon speed grades, so that buying a faster (more expensive) device, may enable a particular design to operate at the required clock rate.

## Overcoming obstacles

In designing a radar pulse compression IP core flexible enough for diverse systems, it soon became clear that the conflicting demands for size, dynamic range and speed would require some clever signal-processing techniques and multiple architectures. Some of the strategies for meeting these objectives for Pentek's GateFlow 4954-440 pulse compression IP core are described. The core is targeted for the Virtex-II, Virtex-II Pro and Spartan device families from Xilinx.

To tackle the dynamic range issue, a major design decision was made to use block floating point arithmetic throughout to achieve some of the accuracy benefits of floating-point math while preserving the reduced size benefits of fixed-point hardware. This technique involves adaptive scaling of all of the points in a vector by the same amount, so that the largest point just fits in the bit field without overflowing.

In practice, all of the output points of a particular signal-processing stage are stored in a RAM. The entire output block (or vector) is then scanned to determine the largest point. Then all of the points in the block are left-shifted by the same number of bits required to left justify the largest point. This number of shifts is then tagged with the block as its exponent and passed on to the next stage.

Three block floating point conversion stages are incorporated in the Core 440 design as shown in Figure 3. With this arrangement, block floating point arithmetic maximizes the dynamic range of a given word length and adaptively scales for changing signal levels automatically after each of the three stages. The output pulse is delivered in block floating point format to preserve accuracy.

To handle varying accuracy requirements under this block floating point scheme, the Core 440 is offered with three different word lengths (mantissa): 16, 20 and 24 bits. The 16-bit version uses a single 18 x 18 multiplier stage while the 20-bit and 24-bit versions use the bulkier compound

multiplier stages described earlier.

The reference pulse spectrum is stored in a RAM array that can be loaded directly through a data port. Instead of loading the spectrum of the reference pulse, an alternate path is provided so that the time domain waveform of the reference pulse can be processed by the input FFT and then sent into the reference pulse spectrum RAM. For fixed modulation patterns for transmitted pulses, the RAM needs to be loaded only once, but for adaptive systems, a new reference pulse spectrum can be loaded for each processing frame.

Another design decision centered on how to support different FFT (and IFFT) sizes or block lengths. Parameter entry of the FFT size is desirable to support multiple modes with a single FPGA design, and 16 k points was chosen as a reasonable maximum length. However, making provisions to support a 16 k point FFT consumes a great deal of the RAM resources, forcing customers with smaller FFT requirements to use a larger and more expensive FPGA than necessary. For this reason, four different maximum length FFT designs were created for the Core 440: 2 k, 4 k, 8 k and 16 k points. In each case, the size of the FFT is programmable from 64 points up to the maximum size in binary steps, simply by entering a parameter in a FPGA register.

One additional architectural option was deemed important. Since the FFT and the IFFT blocks involve nearly identical processing tasks, it is possible to use the same FPGA hardware to perform these two operations sequentially. If the pulse repetition rate is low enough, this can result in a dramatic reduction in the number of FPGA resources.

Accordingly, the Core 440 offers two different architectures. The maximum performance architecture version is shown in Figure 3, with two dedicated engines, one for the FFT and another for the IFFT. The minimum resource architecture, shown in Figure 4, includes switches at the input and output of the FFT/IFFT block to engage the signal flow paths at the appropriate times. The resulting output of both architectures is identical, so users can trade off speed for resource utilization, perhaps allowing the core to fit in a much smaller device or leaving room for additional functions.

## Summary

In all, the Core 440 offers three different bit widths for the processing engines, four different maximum length FFTs, and two different speed/resource architectures for a total of 24 different configurations. This flexibility comes in handy since requirements may shift during the design cycle.

Although FPGA development tools are improving rapidly, nothing replaces the intuition and guidance of an experienced design engineer who uses hardware and software skills to balance performance needs against the physical realities of configurable logic. **DE**


### ABOUT THE AUTHOR

Rodger H. Hosking is vice president of Pentek and was one of the co-founders of the company in 1986. With more than 26 years experience in the electronics industry, he is responsible for matching new technology to advanced signal-processing applications and for the definition of new products. He designed the first commercial direct digital frequency synthesizer, and holds patents in frequency synthesis and FFT spectrum analysis techniques. Hosking has a BS degree in Physics from Allegheny College and BS and MS degrees in Electrical Engineering from Columbia University.