Using Programmable Logic to Accelerate DSP Functions

XILINX®

Steven K. Knapp
Corporate Applications Manager

Xilinx, Inc.	**Xilinx Asia Pacific**
2100 Logic Drive	Unit 2308-2319, Tower 1
San Jose, CA 95124	Metroplaza, Hing Fong Rd.
U.S.A.	Kwai Fong, N.T., HONG KONG
1-408-559-7778 (TEL)	852-2410-2739 (TEL)
1-408-879-4442 (FAX)	852-2494-7159 (FAX)
dsp@xilinx.com (E-mail)	**hongkong@xilinx.com** (E-mail)

http://www.xilinx.com (Web site)

Abstract

Programmable logic offers an alternative solution for the computationally-intensive functions found in Digital Signal Processing (DSP). Programmable logic can provide increased DSP system performance at reduced system cost. Programmable logic combines the flexibility of a general-purpose DSP plus the speed, density, and low cost of an ASIC implementation. In some applications, programmable logic replaces the DSP processor entirely. In others, programmable logic works in conjunction with the DSP processor, offloading the computationally-intensive function and freeing the processor for other functions.

Introduction

Digital Signal Processing (DSP) is one of the fastest-growing applications in the electronics industry. It is used in a wide range of applications including:

- Telecommunications
- Data communications
- Wireless communications
- Image enhancement and processing
- Data acquisition
- Remote sensing
- Radar
- Video processing
- Broadcasting (HDTV)
- Voice synthesis and recognition

While there are many high-performance DSP processors on the market, they are not suited to all DSP applications. Their general-purpose architecture makes these DSP processors flexible, but they may not be fast enough or cost effective for all systems.

This paper describes an alternative to traditional, general-purpose DSP processors. Programmable logic can implement functions beyond the capabilities of today's DSP processors. Field Programmable Gate Arrays (FPGAs) or Complex Programmable Logic Devices (CPLDs) potentially provide performance increases of an order of magnitude over traditional DSPs with the same flexibility.

What is DSP?

Before exploring how programmable logic provides various DSP functions, a broader definition of DSP is required. The term "DSP" applies broadly to continuous mathematical processes, attempted in real-time. These include functions such as:

- Digital Filtering
  - Finite Impulse Response (FIR)
  - Infinite Impulse Response (IIR)
  - Viterbi Decoder
- Convolution
- Correlation
- Fast Fourier Transforms

Most of these functions require the incoming data to be multiplied or added with various internal feedback mechanisms to perform the desired mathematical function. This function is generically called Multiply/Accumulate.

To increase performance, most general-purpose DSP processors perform a multiply/accumulate function in a single clock cycle (or less). The hardware to perform this function is called a

Multiply/Accumulator (MAC). Most DSP processors have a fixed-point MAC while some have a more expensive floating-point MAC.

Traditional Approaches

Traditionally, DSP functions are either implemented in a general-purpose DSP processor or built using ASIC technology.

Generally, ASIC or gate array technology is used whenever the application requires performance beyond the abilities of current DSPs, or when the expected system volumes justify a semi-custom solution.

However, programmable logic provides a third solution that combines the best of both DSP and ASIC technologies without their respective limitations.

The Promise of Programmable Logic

The Best of DSP and ASIC Technologies

Increased Flexibility

Like a general-purpose DSP, FPGAs and CPLDs are programmable and changeable. The designer can make changes quickly without the additional cost and long lead-times of an ASIC. FPGAs, like DSPs, have no minimum volume requirements as do ASICs.

When performance is a factor, many designers turn to ASIC technology. ASIC technology offers the ability to design a custom architecture that is optimized for the target application.

For example, digital filtering typically requires numerous MAC cycles—one MAC cycle for each filter tap. A traditional DSP only has a single MAC, so each filter tap must be executed sequentially. An ASIC implementation of a filter algorithm might have numerous MACs so that the all of the taps can be processed in parallel.

Likewise, a Field Programmable Gate Array (FPGA) has a flexible architecture that can be adapted for a specific DSP function. Also, FPGAs have sufficient capacity to fit multiple MACs or algorithms into a single device along with the interface circuitry required by the application—a single-chip solution compared to a DSP processor.

Increased DSP performance with FPGAs

DSP architecture directly affects system performance. Because most DSP functions are multiply/accumulate-based, the performance of the MAC is crucial.

Nearly every processor is capable of performing DSP algorithms because nearly every processor can perform additions and multiplies. The only difference between a general-purpose DSP and
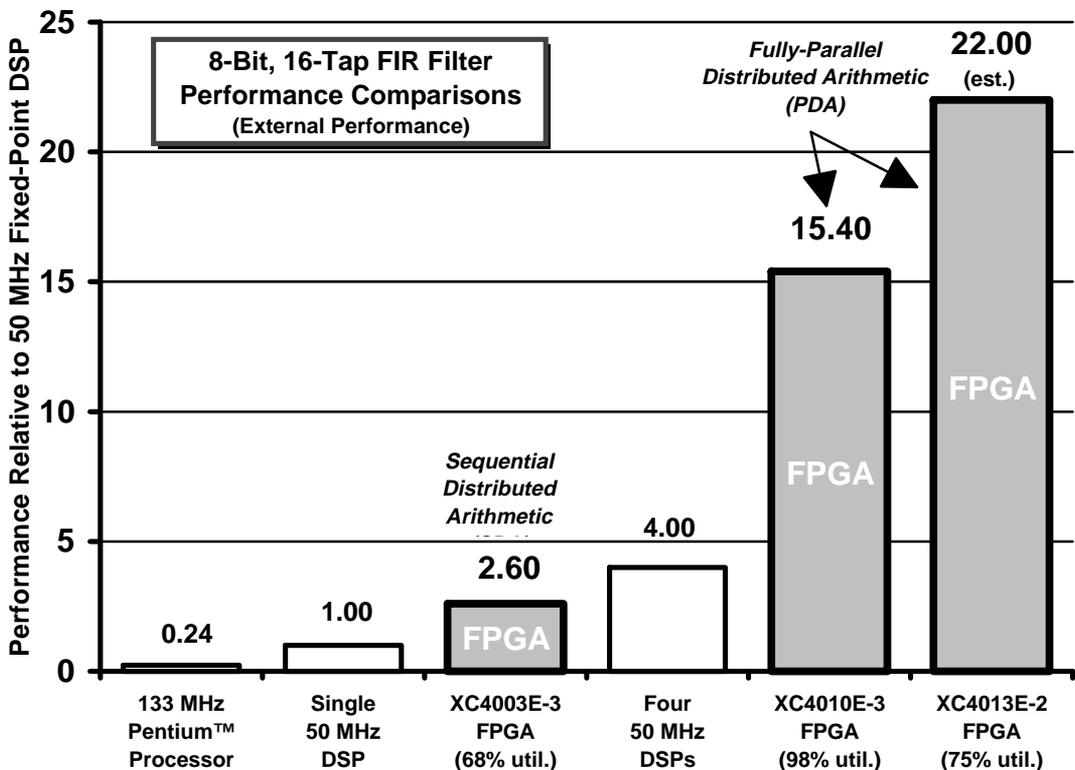

Figure 1. Relative performance for various implementations of an 8-bit, 16-tap FIR filter compared to a 50 MHz fixed-point DSP processor. FPGAs are up to 22 times faster.
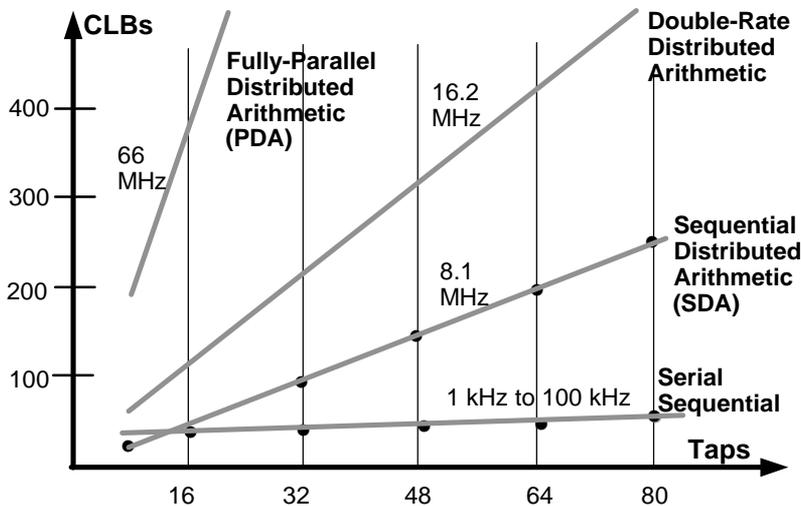
Figure 2. Performance of different Distributed Arithmetic (DA) FIR filter implementations and their relative silicon efficiency in XC4000E FPGA logic blocks (CLBs).

a microprocessor is how well they perform this function.

For example, the Pentium™ processor requires 11 clock cycles to perform a single multiply/accumulate operation whereas most DSP processors require just a single cycle. A 50 MHz fixed-point DSP performs a multiply/accumulate cycle in only 20 ns whereas a 133 MHz Pentium processor requires 1.3 µs to perform the same function. As a result, a 133 MHz Pentium processor has only 24% the DSP processing power of a 50 MHz DSP for the filter function shown in Figure 1.

Each tap of a digital filter requires one MAC cycle. For example, a 16-tap filter requires 16 MAC cycles. Because most DSPs only have a single MAC unit, each tap is processed sequentially, slowing overall system performance.

Some of the more powerful—and correspondingly more expensive—DSPs have multiple MACs. These DSPs perform multiple MACs in one clock cycle. The same goal is accomplished by using multiple single-MAC DSPs with shared high-speed memory. In both cases, extra performance is bought with higher component cost plus board space.

FPGAs offer an even more powerful architecture—one tailored to the specific application. Because the logic in an FPGA is flexible and amorphous, the DSP function can be mapped directly to the resources available on an FPGA.

Not only is the FPGA implementation faster than most DSPs, it offers tradeoffs between system density and performance. Figure 1 shows the relative performance of various implementations of an 8-bit, 16-tap FIR filter, normalized to the performance of a 50 MHz fixed-point DSP processor.

The most efficient FPGA implementation shown uses 68% of an XC4003E-3 FPGA, or roughly 1,500 gates [1]. This implementation outperforms a single 50 MHz DSP by a factor of 2.6. The key to its efficiency is the Sequential Distributed Arithmetic (SDA) algorithm [2, 3]. This algorithm takes advantage of the XC4000E architectural features. The multiply functions are mapped into the FPGA's function generators, the adders and accumulators use the XC4000E fast carry logic, and the serial shift registers are built in efficient, on-chip RAM [4].

The highest performance FPGA implementation uses about 75% of an XC4013E-2 FPGA, or about 9,750 gates. Though roughly seven times larger than the space-efficient version, the high-performance implementation is 22 times faster than a 50 MHz DSP for this application. It uses a Parallel Distributed Arithmetic (PDA) algorithm [2, 3]. Even higher performance is possible if the application can tolerate the extra data latency caused by pipelining. Performance will also be higher if the filter is integrated with other logic in the same chip thereby bypassing I/O delays.

A broad range of alternate FPGA implementations is possible. The trade-offs between density and performance for various algorithms is shown in Figure 2. Each implementation is tailored to the speed, density, and cost requirements of the target application. Serial sequential is the most efficient, but also the slowest. PDA is the fastest, but also uses the most logic. SDA is a good compromise of speed and density, depending on system requirements.
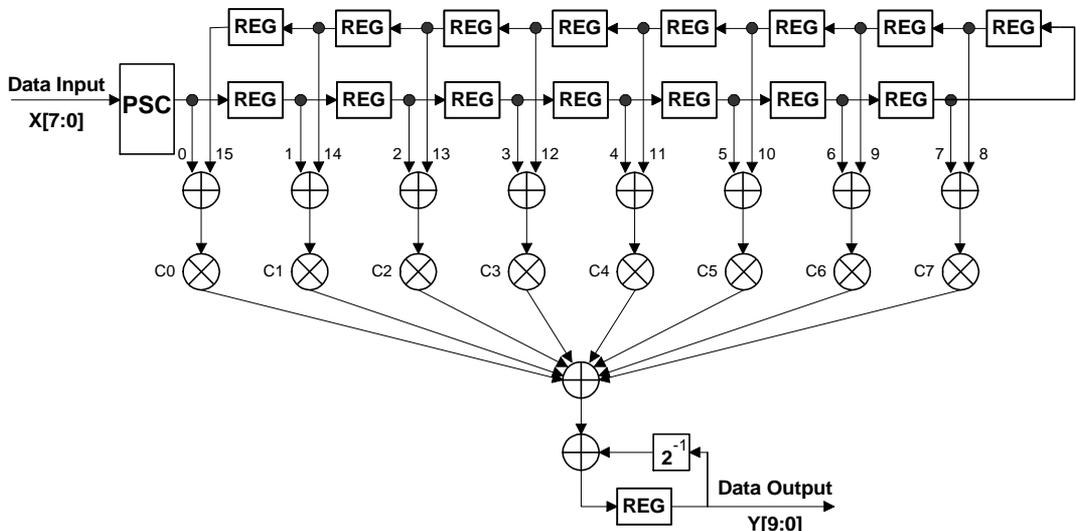
Figure 3. Data flow diagram for a 16-tap FIR filter showing the parallel operations and feedback paths that slow DSP processor performance. FPGAs perform multiple operations per clock cycle, resulting in better performance.

Programmable Logic Reduces DSP Cost

FPGA Replaces DSP Processor

In some applications, a single FPGA or CPLD completely replaces the dedicated DSP. These applications are typically embedded processing or filtering functions with data sample rates between 100 kHz up to 70 MHz.

In embedded filtering applications, it is unlikely that the DSP functionality will change, therefore the extra flexibility of a general-purpose DSP adds costs without benefit.

In the 1 kHz to 100 kHz range, the DSP function—plus all other system logic—fit in a single, low-cost FPGA. This approach uses a silicon-efficient, but low-performance, serial-sequential algorithm as shown in Figure 2.

FPGA Enhances General-Purpose DSP

FPGAs and CPLDs will never completely replace general-purpose DSP processors. Current-generation programmable logic addresses the fixed-point DSP portion of the market. General-purpose DSPs still dominate in floating-point performance. Also, general-purpose DSP processors utilize familiar software methods. The designer implements the DSP algorithm using a programming language like 'C' and compiles the code for a specific DSP processor.

In many applications, a fast and very expensive DSP processor is used to handle the peak performance of a small piece of code. A typical DSP algorithm contains many repetitious feedback loops and parallel structures as shown in the data flow diagram for the 16-tap FIR filter in Figure 3. The software code for such algorithms is not efficiently implemented in general-purpose DSP architectures. Typically, about 20–40% of the DSP's code utilizes 60–80% of the DSP's processing power.

One popular method to boost DSP performance is to use multiple DSPs in parallel plus high-speed memory. For example, the four-DSP solution shown in Figure 1 is theoretically four times the performance of the single-DSP solution. However, the cost is more than four times higher. These multi-chip DSP designs generally require more board space and higher performance memories that adds cost.

Instead, the best solution for these applications may be a DSP processor, microprocessor, or micro-controller with an FPGA co-processor. The general-purpose DSP processor handles the system control and data movement functions. The FPGA provides a custom-tailored DSP co-processor to handle the peak processing function. See ***Case Study: Viterbi Decoder*** for a more specific example.

Analyzing the DSP algorithm will reveal any parallel structures and iterative loops that consume DSP processing power. Placing these functions in the FPGA enhances the overall performance.

The FPGA-based DSP accelerator concept is similar to a floating-point co-processor working with a microprocessor. Recognizing this potential, Xilinx developed the SRAM-based XC6200 FPGA family designed specifically for co-processing in embedded system applications as shown in Figure 4 [5].

The XC6200 consists of:

- A fast, high-density FPGA architecture

- An efficient FPGA architecture for arithmetic and data path applications

- An integrated, high-speed processor interface

- A fast configuration and partial reconfiguration
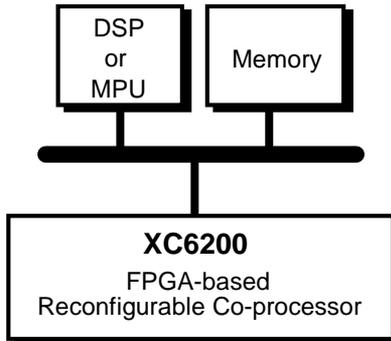
- Easy access to internal logic and flip-flops

Figure 4. XC6200 FPGA accelerates processing applications through a high-speed processor interface.

FPGA Instead of ASIC

FPGAs also replace ASICs in DSP systems. Designers chose ASIC in the past for two reasons: Either they required DSP processing power beyond the capabilities of a general-purpose DSP, or the system had sufficiently high production volumes to justify a semi-custom solution.

Like ASICs, FPGAs and CPLDs can provide superior performance to general-purpose DSPs. Before high-density FPGAs were available, companies that required DSP performance but lacked volume system shipments were often forced into ASIC technology. The minimum volume requirements plus the long lead-times, non-recurring engineering changes (NRE), and risk of an ASIC are unacceptable for low-volume project. FPGAs provide the performance and architectural flexibility of ASICs but are user-programmable for low development costs.

Contrary to popular belief, FPGAs also offer a solution for high-volume designs. Plus, Xilinx provides an additional option for the high-volume user. Xilinx HardWire™ gate arrays are 100% pin- and functionally-compatible with a corresponding Xilinx FPGA and reduce component cost up to 50%–80%. The HardWire gate arrays use the same netlist, the same layout database, and the same silicon fabrication facility. Consequently, the engineer only designs the application once—Xilinx converts the design and provides the test vectors for 100% fault coverage.

Design debugging, system verification, and initial production are done with FPGAs. Once verified, the HardWire gate arrays provide a low-risk migration path to a high-volume, low-cost solution.

An additional benefit of SRAM-based FPGAs over ASICs is that they can be reprogrammed, on the fly, in the system. Consequently, a single FPGA can implement different DSP functions at various times in a system to boost overall performance.

Case Study: Viterbi Decoder

A company developed a DSP-based telecommunications system. One of the key DSP algorithms was a Viterbi decoder used as part of a noise-cancellation circuit [6]. The initial design used two 66 MHz general-purpose DSP processors that have just recently become available. High-speed SRAM memory was also required to meet the performance goals of the algorithm and the system.

Though a Viterbi decoder does not require any multiply operations, it can still be considered a DSP algorithm because of its mathematical processing. The algorithm, shown graphically in Figure 5, required 17 computational clock cycles, plus an additional 7 clock cycles due to wait-states for the DSP's external SRAM memory. The seven 24-bit (2's-complement) data words are multiplexed together on a common 33 MHz I/O bus. As a result, the Viterbi decoder algorithm required 360 ns processing time (24 cycles at 15 ns per cycle) and consumed about 80% of the combined DSP's overall processing time.

There are two limiting factors in this DSP-based design. First, the external SRAM timing required an extra 15 ns wait-state, limiting the data bus to 30 ns for each transaction. Second, each Add/Subtract and Multiplex stage must be performed sequentially in the DSPs. The Add/Subtract stages each required four separate operations with multiple instructions.

This algorithm is well suited for the FPGA. The FPGA's ability to process parallel data paths accommodates the parallel structures of the four ADD/SUB-blocks in the first stage and the two SUB-blocks in the second stage. The two MUX-blocks take advantage of the ability to register and hold the input data until needed with no additional clock cycles.
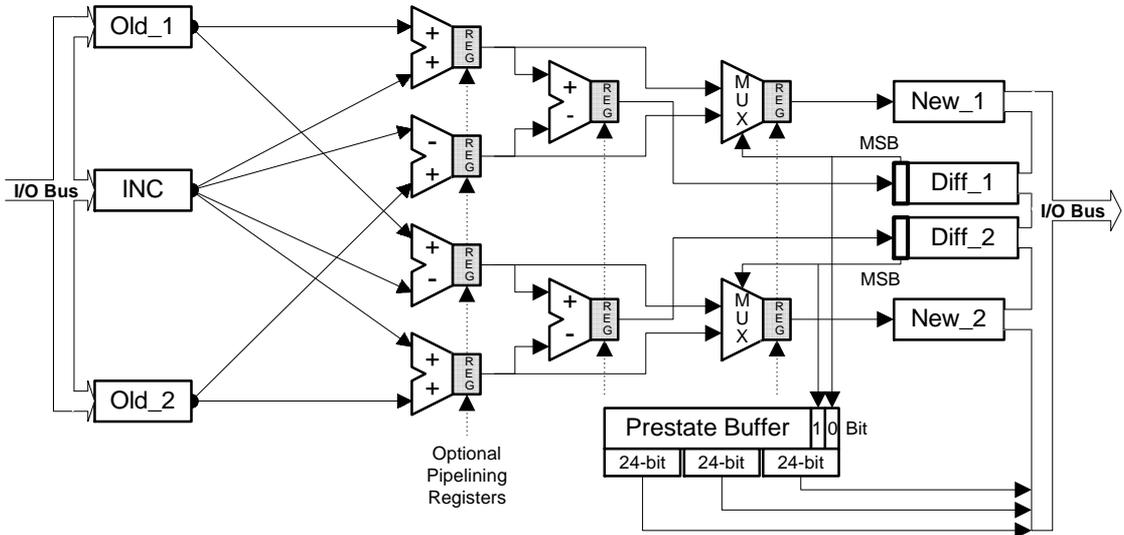
Figure 5. Viterbi decoder block diagram. FPGAs implement the multiple Add/Subtract functions in parallel, resulting in superior performance.

The design conversion resulted in faster performance as shown in Figure 6. The FPGA-based Viterbi decoder cycle time is 135 ns compared to 360 ns by the dual-DSP design—a 62% improvement. The I/O data bus also supports the full 66 MHz bandwidth supported by the DSP processor—twice the original throughput. The FPGA-based implementation replaced one of the programmable DSPs and three SRAM chips as shown in Table 1, resulting in significantly higher performance and lower system complexity. The Viterbi decoder consumed 44% of the XC4013E-3 FPGA. The remaining space was filled with other system logic.

Table 1. Reduced Parts Count with FPGA.

DSP-Only	DSP + FPGA
8 DEVICES	**4 DEVICES**
Two 66 MHz DSPs	One 66 MHz DSP
Six 15 ns SRAMs	XC4013E-3 FPGA (44%)
System logic	Three 15 ns SRAMs

This design can also be implemented with the original 33 MHz I/O bus performance. This implementation minimizes the CLBs required by exploiting the symmetrical nature of the design. Note that the `Old_1` and `Old_2` values follow the same path with only a minor difference in the magnitude of the second stage SUB-block. This implementation requires the data on the I/O data bus to be written and read in a specific order.

Summary: Using FPGAs for DSP

The previous case study is just one example of how FPGAs accelerate DSP performance and how they reduce overall system cost.

Finding the Right Function

To use FPGAs to enhance the performance of a DSP application:

- *Identify the parallel data paths* in the algorithm. The FPGA can implement these functions in parallel. A DSP must execute these sequentially.

- *Find operations that require multiple clock cycles* when executed in a general-purpose DSP. Again, take advantage of the FPGA's parallelism.

DSP Functions That FPGAs Do Best

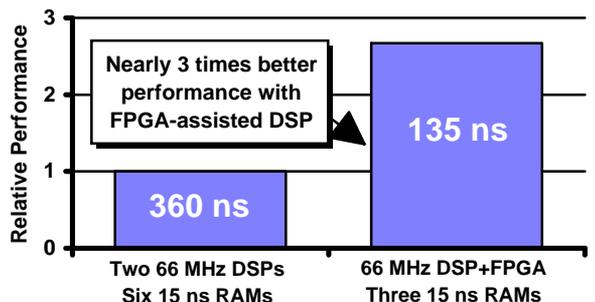Other DSP applications that benefit from FPGA technology include those requiring:

Figure 6. Performance of two Viterbi decoder implementations. The DSP+FPGA solution is faster.

- **High sample rates** — FPGA-based DSP systems outperform one or more general-purpose DSP processors as shown in Figure 7.

- **Low sample rates** — At data rates between 1 kHz to 100 kHz, the DSP function can be easily integrated along with other system logic in a low-cost FPGA using a very efficient serial sequential algorithm.

- **Short word length** — FPGA-based DSP designs run faster as the word width decreases when using the space-efficient SDA algorithm.

- **Lots of filter taps** — The number of filter taps has little effect on an FPGA-based DSP design when using the space efficient SDA algorithm.

- **Single-chip solution required** — Integrate the DSP function and all of the system logic in a single FPGA.

- **Fast correlators** — The look-up table architecture of Xilinx FPGAs provides a fast and efficient way to build correlators.

- **Low-cost migration path** — Xilinx HardWire gate arrays provide a low-risk, 100% pin- and functionally-compatible migration path to a high-volume, low-cost production solution. No simulation, test vectors, or re-engineering required.

Other Resources

DSP Applications Group

For DSP applications information or for help on implementing an algorithm in programmable logic, contact the Xilinx DSP Applications Group via E-mail at **dsp@xilinx.com** or via FAX at 1-408-879-4442.

Xilinx WebLINX DSP Site

Xilinx has a home page on the World-Wide Web that includes a special section for DSP. The WebLINX UURL for DSP is

http://www.xilinx.com/appsweb.htm#DSP

Application notes and data sheets are also available via WebLINX.

References

[1] Goslin, G. R. & Newgard, B. "16-Tap, 8-Bit FIR Filter Application Guide," Xilinx, Inc., November, 1994.
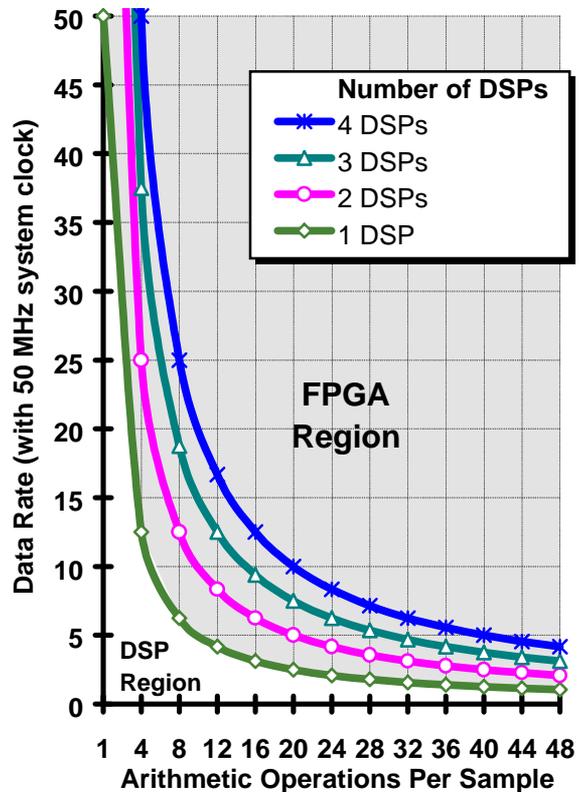
Figure 7. Graph showing the performance of FPGA-based DSP vs. one or more DSP processors. Shaded region indicates where FPGA is a better solution.

Available on WebLINX and via E-mail. Send an E-mail message to **xdocs@xilinx.com** with **send 80000** in the Subject header.

[2] New, Bernie. "A distributed arithmetic approach to designing scaleable DSP chips," **EDN**, August 17, 1995, pp. 107-114.

[3] Goslin, G. R. "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices," **Proceedings of the 1995 DSP^X Technical Program**, pp. 595-604.

[4] Xilinx, Inc., "XC4000E Field Programmable Gate Array Family Product Specification," Version 1.04, September, 1995.

Available on WebLINX and via E-mail. Send an E-mail message to **xdocs@xilinx.com** with **send 80002** in the Subject header.

[5] Xilinx, Inc., "The XC6200 FPGA Family, Advance Product Information," August, 1995, P/N: 0010259-01.

[6] Viterbi, A. J. & Omura, J. K., **Principles of Digital Communication and Coding**, McGraw-Hill, New York, 1965.

Glossary of Terms

ASIC—Application-Specific Integrated Circuit, commonly called a gate array.

CPLD—Complex Programmable Logic Device. Also called EPLD or Erasable Programmable Logic Device.

DA—Distributed Arithmetic. An alternate approach to implementing arithmetic functions.

DSP—Digital Signal Processing

FIR—Finite-Impulse Response. A type of digital filter.

FPGA—Field Programmable Gate Array.

HDL—Hardware Description Language such as VHDL and Verilog.

IIR—Infinite-Impulse Response. A type of digital filter.

MAC—Multiply/Accumulator. A DSP processor provides good performance in DSP applications because it executes a multiply and an addition in a MAC unit in a single clock cycle.

NRE—Non-Recurring Engineering. The set-up or mask charges required to build an ASIC.

PDA—Parallel Distributed Arithmetic. An alternate approach to implementing arithmetic functions. Efficient and high performance in some FPGA architectures.

PSC—Parallel to Serial Converter.

SDA—Sequential Distributed Arithmetic. An alternate approach to implementing arithmetic functions. Very efficient and a good compromise between speed and density in some FPGA architectures.