A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance

Gregory Ray Goslin
Digital Signal Processing Program Manager
Xilinx, Inc.
2100 Logic Dr.
San Jose, CA 95124

Abstract:

FPGAs have become a competitive alternative for high performance DSP applications, previously dominated by general purpose DSP and ASIC devices. This paper describes the benefits of using an FPGA as a DSP Co-processor, as well as, a stand-alone DSP Engine. Two Case Studies, a Viterbi Decoder and a 16-Tap FIR Filter are used to illustrate how the FPGA can radically accelerate system performance and reduce component count in a DSP application. Finally, different implementation techniques for reducing hardware requirements and increasing performance are described in detail.

Introduction:

Traditionally, digital signal processing (DSP) algorithms are implemented using general-purpose (programmable) DSP chips for low-rate applications, or special-purpose (fixed function) DSP chip-sets and application-specific integrated circuits (ASICs) for higher rates. Advancements in Field Programmable Gate Arrays (FPGAs) provide new options for DSP design engineers. The FPGA maintains the advantages of custom functionality like an ASIC while avoiding the high development costs and the inability to make design modifications after production. The FPGA also adds design flexibility and adaptability with optimal device utilization while conserving both board space and system power, which is often not the case with DSP chips. When a design demands the use of a DSP, or time-to-market is critical, or design adaptability is crucial, then the FPGA may offer a better solution.

The SRAM-based FPGA is well suited for arithmetic, including Multiply & Accumulate (MAC) intensive DSP functions. A wide range of arithmetic functions (such as Fast Fourier Transform's (FFT's), convolutions, and other filtering algorithms) can be integrated with surrounding peripheral circuitry. The FPGA can also be reconfigured on-the-fly to perform one of many system-level functions.

When building a DSP system in an FPGA, the design can take advantage of parallel structures and arithmetic algorithms to minimize resources and exceed the performance of single or multiple general-purpose DSP devices. Distributed Arithmetic[1] for array multiplication in an FPGA is one way to increase data bandwidth and throughput by several order of magnitudes over off-the-shelf DSP solutions. One example is a 16-Tap, 8-Bit Fixed Point, Finite Impulse Response (FIR) filter[2].

The FIR design supports more than 8 million samples per second. This example can also be implemented using multiple bits, until a "Fully Parallel Distributed Arithmetic" algorithm is obtained for higher sample rates (i.e., 55.89 million samples per second).

Figure 1 compares the 16-Tap FIR filter implemented in a state-of-the-art fixed-point DSP with that of the Xilinx FPGA. As published by Forward Concepts[5], "For 1995, the state-of-the-art fixed-point DSP is rated at 50 MIPS." Such a device requires 20 nsec per Tap to implement a 16-Tap FIR filter, which translates to a theoretical maximum (with zero wait-states) sample rate of 3.125 million samples per second.

An In-System Programmable (ISP) FPGA can also be reconfigured on the board during system operation. Taking advantage of the reconfigurability feature means a minimal chip solution can be transformed to perform multiple functions. For example, an FPGA could be the basis for a system that performs one of several DSP functions. Suppose, for instance, one function is to compress a data stream in transmit mode and another function is to decompress the data in receive mode. The FPGA can be reconfigured on-the-fly to switch, or toggle, from one function to another. This capability of the FPGA adds functionality and processing power to a minimum-chip DSP system controlled with an internal or an external controller. This "Reconfigurable Computing" technique is beginning to impact design methodologies.
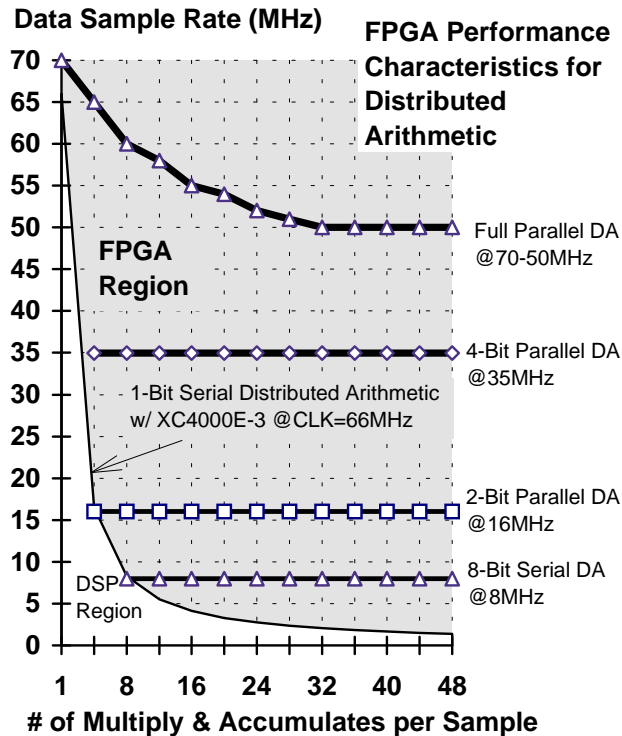
Data Sample Rate (MHz)

Figure 1.   Distributed Arithmetic Performance for 16-Tap, 8-Bit Fixed Point, FIR Filter.

The FPGA design cycle requires less hardware-specific knowledge than most DSP chips or ASIC design solutions. Smaller design groups, with less experienced engineers, can design larger, more complex DSP systems in less time than larger design groups with more experienced engineers who are required to know device-specific programming languages. The FPGA-based DSP system-level design team can design, test, verify, and ready a complex DSP system for production in weeks.

FPGA-Based DSP Accelerator:

General-purpose DSP devices have been the enabling technology for most mid to high-end electronic systems. Although high-volume applications favor ASIC implementations, the general-purpose DSP devices are used to drive the technology forward.

The engine in the DSP system is typically a specialized time-multiplexed sequential computer system that performs a continuous mathematical process, attempted in real-time [see Figure 2]. While the DSP processor may perform multiple instructions per clock cycle (Harvard Architecture), the overall process is performed in a three-step series of 1⇒memory-read, 2⇒process, and 3⇒memory-write instructions. This process is adequate for independent sequential algorithms (a modified Von Neumann Architecture). The DSP

processor becomes less efficient when an algorithm is dependent on two or more of the past, present, and/or future state conditions. This is primarily due to the feed-back or parallel structure of the data flow being processed sequential with additional wait-states in a DSP.
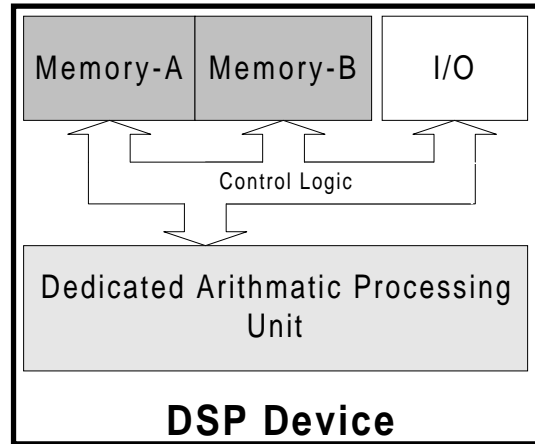

Figure 2.   Basic DSP Block Diagram

System performance requirements continue to increase. The advancement in performance for DSP devices is lagging behind this growth. Because of this imbalance the DSP designer is forced to use a systolic array of DSP processors to boost the overall performance.

A typical DSP algorithm contains many feed-back loops or parallel structures [see Figure 3]. The software code for a DSP algorithm of this type is not efficiently implemented in a general purpose DSP. Typically, about 10-30% of the DSP code utilizes 60-80% of the processors power. Analyzing the DSP algorithm and breaking out any parallel structures or repetitive loops into multiple data paths, one can enhance the overall performance of the algorithm. The multi-path parallel data structures can be processed either through parallel DSP devices or in a single FPGA-based DSP hardware accelerator with or without the assistance of a DSP device.

The FPGA is well suited for many DSP algorithms and functional routines. The FPGA can be programmed to perform any number of parallel paths. These operational data paths can consist of any combination of simple and complex functions, such as; Adders, Barrel Shifters, Counters, Multiply and Accumulation, Comparators and Correlators just to mention a few. The FPGA can also be partially or completely reconfigured in the system for a modified or completely different algorithm. (Note that the Xilinx XC6200[6] family can be partially reconfigured while the rest of the device is still active in the system.) The primary concept is to unload the compute-intensive functions requiring multiple DSP

clock cycles into the FPGA and allow the DSP processor to concentrate on optimized single-clock functions.
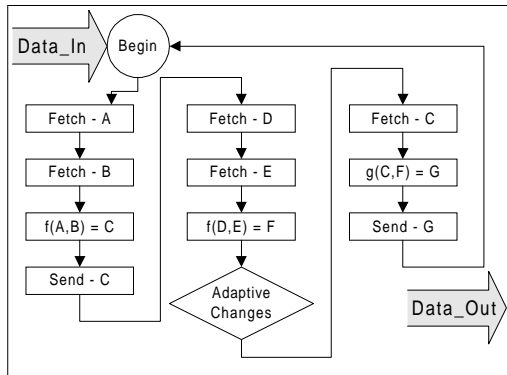

Figure 3.   Basic DSP Algorithm Process

The FPGA-based DSP accelerator is conceptually similar to a Coprocessor for the Microprocessor Units (MPUs) in the computer market. Initially, the MPU required a Math-Coprocessor to accelerate computational algorithms. The functionality of the Coprocessor became so frequently used that it is now an integrated part of the latest processor families. Some DSPs have been optimized to do some dedicated functions much faster than a MPU (Von Neumann architecture). For example, some DSPs can Multiply and Accumulate (MAC) in one clock cycle (DSP @66 MHz = 15 nsec per MAC) compared to eleven clock cycles (P5@100 MHz = 110 nsec per MAC) in the Pentium™ processor.

The combined functionality of the FPGA and the general-purpose DSP can support several magnitudes higher data throughput than two or more parallel DSP devices. The FPGA/DSP implementation is more flexible and proves to be more cost effective than multiple DSPs or an ASIC.

Case Study

Viterbi Decoder

A Viterbi Decoder[7] is a good example of how the FPGA can accelerate a function [Refer to Figure 4]. The initial design used two 66 MHz programmable DSP devices to implement the algorithm. The algorithm used to calculate the "New_1" and "New_2" outputs, shown in Figure 4, required 17-computational clock cycles, plus an additional 7-clock cycles due to the wait-state associated with the DSP's external SRAM memory. This memory was required by the programmable DSP for data storage. Hence, the Viterbi Decoder algorithm required 360 nsec [(24-clock cycles)•(15 nsec)] of processing time. The Viterbi Decoder algorithm utilized

about 80% of the overall processing time. Note this did not include the calculation of the "Diff_1" and "Diff_2" outputs. These two outputs would have required an additional seven clock cycles. The seven (2's-Complement) I/O Data words were multiplexed on a common I/O Bus at the maximum I/O rate of 33 MHz.
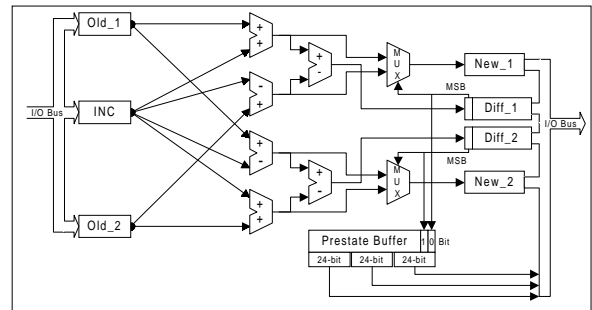

Figure 4.   Viterbi Decoder Block Diagram

There were two limiting factors for this DSP-based design. First, the wait state associated with the DSP's external SRAM memory required two 15 nsec clock cycles for each memory access. Hence, the data Bus transfer required 30 nsec for each data transaction. This forced the I/O Bus speed to a maximum of 30 nsec. Secondly, each Add/Subtract and MUX stage had to be performed sequentially with additional wait-states. The Add/Subtract stages required four additional operations with multiple instructions.

FPGA & DSP Based Viterbi Decoder:

The Viterbi Decoder is well suited for the FPGA. The ability to process parallel data paths within the FPGA takes advantage of the parallel structures of the four Add/Subtract-blocks in the first stage and the two Subtract-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 external memory or additional clock cycles.
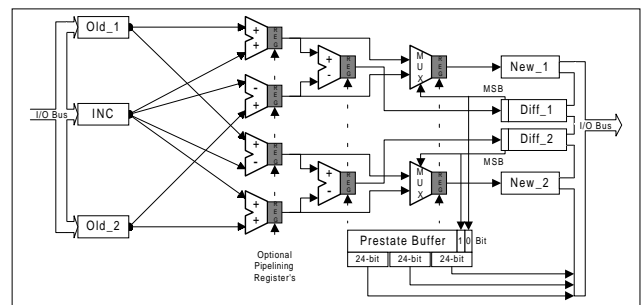

Figure 5.   FPGA-Based Viterbi Decoder Block Diagram

The design conversion resulted in the following performance: The FPGA based Viterbi Decoder required 135 nsec [(9-clock cycles)•(15 nsec)] of total

processing time (this includes all of the outputs) compared to the 360 nsec required for the partial output data by the DSP. This enhancement equates to 37.5% of the original DSP processing time or 62.5% better processing performance. The I/O Data Bus can also support a 66 MHz data transfer rate, supporting twice the original throughput. The FPGA-based implementation replaced a programmable DSP and three SRAM chips.
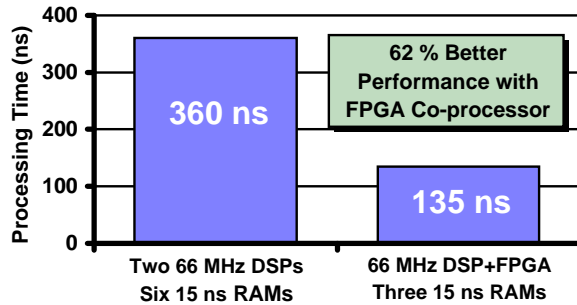

Figure 6. Performance of two Viterbi decoder implementations. The DSP+FPGA solution is 2.7 times faster

This design can also be implemented in a multiplexed version with the original 33 MHz I/O Data transfer rate. This multiplexed implementation minimizes the CLB resources used. This is done by observing the symmetrical nature of the design. Note that "Old_1" and "Old_2" Datum follow the same path with only a minor difference in the magnitude of the second stage SUB-block. This implementation requires a specific ordering for data accesses on the I/O Data Bus.

This is one example of how the FPGA can accelerate a DSP function. To use the FPGA in a DSP design, identify the parallel data paths and/or the operations requiring multiple clock cycles in the DSP.

Digital Filter Design:

The processing engine of most filter algorithms is a Multiply and Accumulate (MAC) function. Filter designs can vary over a wide range in the number of MACs, from one to thousands. As the number of MACs increase, the algorithm becomes much more complex for a CPU-based architecture. Hence, the algorithm becomes more compute-intensive for any conventional DSP.

The MAC function can be implemented more efficiently with various Distributed Arithmetic (DA)[1] techniques then with conventional arithmetic methods. Distributed Arithmetic can make extensive use of look-up tables (LUTs), which makes it ideal for implementing DSP functions in LUT-based FPGAs.

FPGA Based Filters:

When building a Digital Filter in an FPGA, the design can take advantage of parallel structures and Distributed Arithmetic algorithms to exceed the performance of multiple general-purpose DSP devices. While the FPGA has no dedicated multiplier, the use of Distributed Arithmetic for array multiplication in an FPGA is one technique used to implement and increase the function's data bandwidth and throughput by several order of magnitudes over off-the-shelf DSP solutions.

One example is a "Serial Distributed Arithmetic" (SDA), FPGA-Based, 16-Tap 8-Bit Finite Impulse Response (FIR) filter[2]. The SDA-FIR design supports an on-off chip I/O-data rate at more than 8 million samples per second [13.7 nsec x (8-Bits + 1) / 16-Taps = 7.71 nsec per Tap] while occupying 68 Configurable Logic Blocks (CLBs)[3] in an XC4000E-3 FPGA. Note that increasing the number of Multiply and Accumulation blocks or Taps has no significant impact on the sample rate when Serial Distributed Arithmetic is used [see Figure 1].
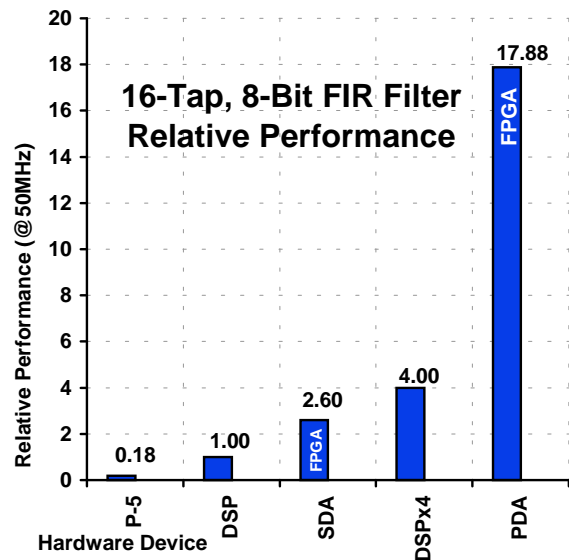

Figure 7. Relative performance for various implementations of a 16-Tap, 8-Bit FIR filter compared to a 50 MHz fixed-point DSP processor

This example can also be implemented using multiple bits, up to a full "Parallel Distributed Arithmetic" (PDA) algorithm to obtain higher I/O-data sample rates which can exceed 55 million samples per second [17.9 nsec / 16-Taps = 1.12 nsec per Tap] and occupies 400 CLBs in an XC4000E-3. Note that these two examples will work

with any 16-Tap configuration with signed or unsigned, fixed point 8-Bit data and coefficients. Optimizing the design for a specific configuration can often reduce the number of CLBs by as much as 20%.

Compare the same 16-Tap FIR filter algorithm implemented in various state-of-the-art fixed-point DSPs with that of the Xilinx FPGA. A fixed-point DSP rated at 50 MIPS requires at least 20 nsec per Tap to implement a 16-Tap FIR filter, which translates to a maximum sample rate of 3.125 million samples per second or 12.5 million samples per second with a 4xDSP multi-chip module, as shown in Figure 8. The performance of general-purpose DSP devices degrades significantly with each additional MAC (one additional Tap), as shown in Figure 8. If the system design requires a systolic array or uses more than one traditional DSP device, consider the performance and cost advantages available using an FPGA.
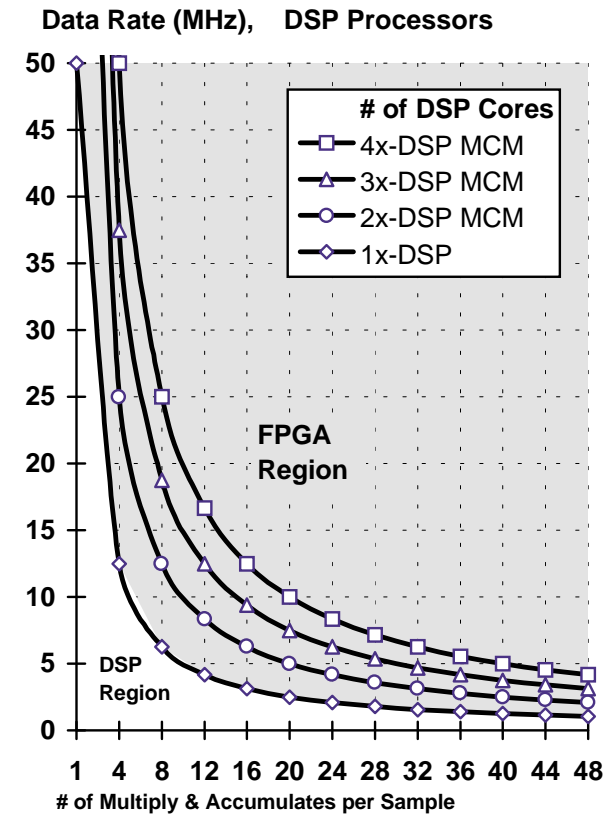

Figure 8. Programmable DSP Performance

Case Study

16-Tap, 8-Bit FIR Filter:

The 16-Tap FIR is a discrete-time filter in which the output is an explicit function only of the present and previous inputs to compute the weighted average of the 16-data sample points. Since the FIR response, mathematically, contains only feed-forward terms, the FIR is unconditionally stable and can be designed to exhibit a linear phase response.
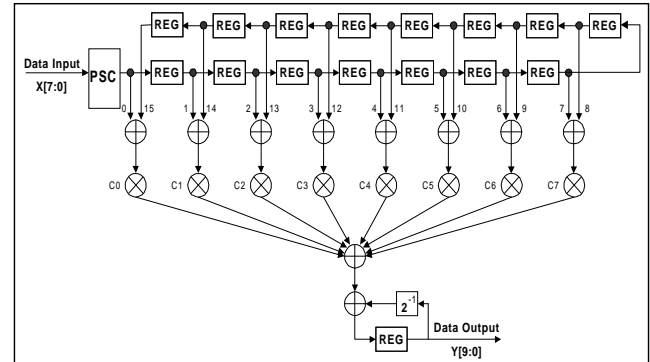

Figure 9.   16-Tap FIR filter Data Flow Block Diagram with Symmetrical Coefficients

FPGA Based 16-Tap FIR Filter:

The FPGA has the ability to implement a FIR filter function using one of several Distributed Arithmetic techniques, depending on the performance required. Note that these techniques can be used to optimize the implementation of many other types of data processing or MAC-based algorithms. "Parallel" Distributed Arithmetic techniques are used to achieve the fastest sample rates, while lower rates can be sustained with a "Serial" or "Serial-Sequential" Distributed Arithmetic techniques that use less resources (fewer CLBs).

The primary design concern is the performance or the sample rate of the filter. The design must work at the sample rate. A design which runs below the sample rate is of no value, while any additional performance, which uses more CLBs, is of no added value.

Distributed Arithmetic:

Distributed Arithmetic differs from conventional arithmetic only in the order in which it performs operations.

Take for example a four-product MAC function that uses a conventional sequential shift-and-add technique to multiply four pairs of numbers and sum the results [see Figure 10].
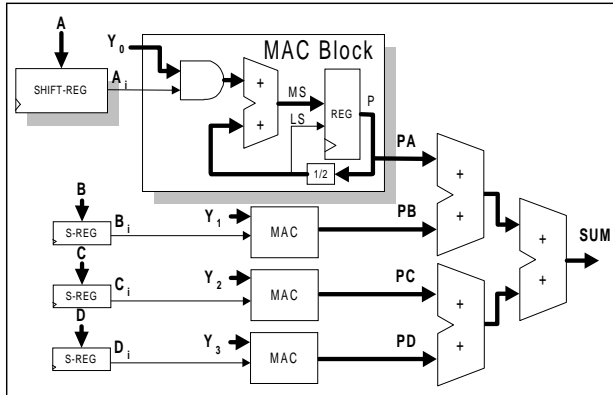
Figure 10. **Conventional four-product MAC using shift-and-add technique to multiply pairs and sum the result**

Each multiplier forms partial products by multiplying the coefficient-Y by 1-bit of the data-A at a time in an AND operation. This technique then adds these partial products into an accumulator that shifts the accumulator's feedback 1-bit position, to the right, to perform a divide by two ($\div 2$) each clock cycle, thus compensating for the bit-weighting of the ingressing partial product [see Figure 11].
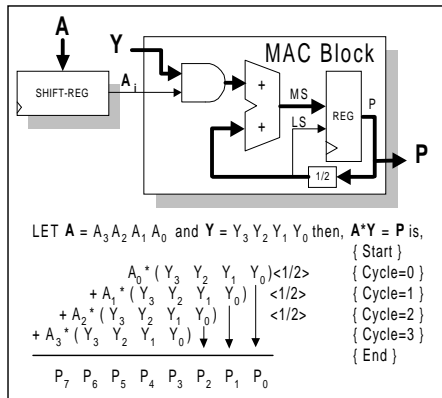

Figure 11. **Four-Bit MAC using shift-and-add technique to multiply**

The four-multiplications are performed simultaneously and the results are then summed when the products are complete. This process requires n-clock cycles for data sample of n-bits. Hence, the processing clock rate is equal to the data rate divided by the number of data bits. During each data clock-cycle, the four-multipliers simultaneously create four-product terms [i.e., in Figure 10; PA, PB, PC, & PD], that eventually are summed into the output. Distributed Arithmetic differs from this process by adding the partial-products before, rather than after, the bit-weighted accumulation.
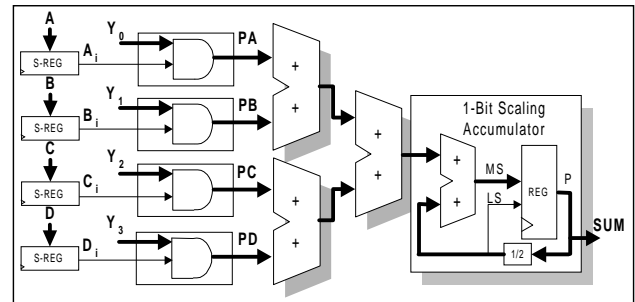

Figure 12. **Serial Distributed Arithmetic for a four-product MAC.**

Using Distributed Arithmetic, as shown in Figure-12, the operations are reordered. This technique reduces the number of shift-and-add circuits to one, but does not change the number of simple adders.

The coefficients in many filtering applications are constants. Consequently, the output of the AND functions and the three adders of Figure 12 depend only on the four input bits from the shift registers. Replacing these AND functions and the three adders with a simple 4-bit (16-word) Look-up Table (LUT) gives the final reduced form of a bit Serial Distributed Arithmetic MAC [see Figure 13].
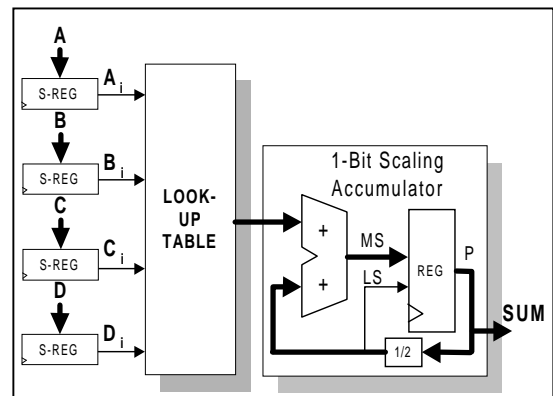

Figure 13. **LUT-Based Serial Distributed Arithmetic for a four-product MAC.**

The LUT data, referenced in Figure 14, is composed of all partial sums of the coefficients (Y_0, Y_1, Y_2, & Y_3). The least significant bit (the output from each serial shift register) of the 4-data samples addresses the LUT. If all four data bits are 1, then the output from the LUT is the sum of all four coefficients. If any data bit is a zero, then the corresponding coefficient is eliminated from the sum. Because the address of the LUT contains all possible combinations of one or zero, based on the four inputs, the LUT output contains all 16 possible sums of the coefficients [see Figure 14].
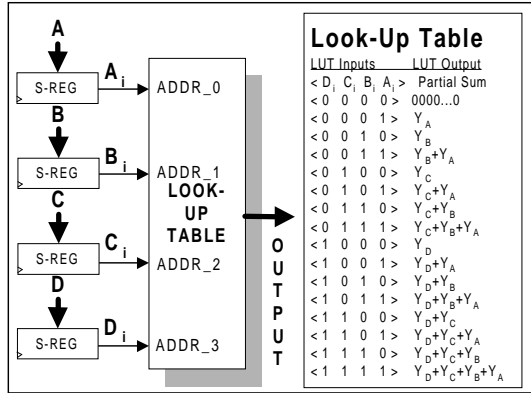
Look-Up Table

LUT Inputs	LUT Output
$< D_i \ C_i \ B_i \ A_i >$	Partial Sum
$< 0 \ 0 \ 0 \ 0 >$	$0000...0$
$< 0 \ 0 \ 0 \ 1 >$	Y_A
$< 0 \ 0 \ 1 \ 0 >$	Y_B
$< 0 \ 0 \ 1 \ 1 >$	$Y_B + Y_A$
$< 0 \ 1 \ 0 \ 0 >$	Y_C
$< 0 \ 1 \ 0 \ 1 >$	$Y_C + Y_A$
$< 0 \ 1 \ 1 \ 0 >$	$Y_C + Y_B$
$< 0 \ 1 \ 1 \ 1 >$	$Y_C + Y_B + Y_A$
$< 1 \ 0 \ 0 \ 0 >$	Y_D
$< 1 \ 0 \ 0 \ 1 >$	$Y_D + Y_A$
$< 1 \ 0 \ 1 \ 0 >$	$Y_D + Y_B$
$< 1 \ 0 \ 1 \ 1 >$	$Y_D + Y_B + Y_A$
$< 1 \ 1 \ 0 \ 0 >$	$Y_D + Y_C$
$< 1 \ 1 \ 0 \ 1 >$	$Y_D + Y_C + Y_A$
$< 1 \ 1 \ 1 \ 0 >$	$Y_D + Y_C + Y_B$
$< 1 \ 1 \ 1 \ 1 >$	$Y_D + Y_C + Y_B + Y_A$

Figure 14.   Contents of a 16-Word Look-Up Table, LUT.

In the four-MAC example, the width of the LUT is typically 2-bits greater than the coefficient width. The additional two-bits allows for word growth from the addition of the four coefficients. The circuit will require at most 2-bits because the circuit is summing no more than four coefficients. Fewer bits may be adequate in cases in which the combinations of coefficients result in less word growth. More than four coefficients require more width for word growth. In general, the number of additional bits should be at least Log_2(number of Coefficients). You can use fewer bits only when the specific coefficients are known and the LUT has been computed.

As the number of coefficients increase, the size of the LUT grows exponentially. A large LUT can be avoided by partitioning the circuit into smaller groups and combining the LUT outputs with adders. The adders are less costly than the larger LUT.

In the example of the 16-Tap FIR filter, the circuit would require a 16-bit LUT. If the FPGA architecture uses four-input function generators, the optimum partition is four products per LUT. The 16-MAC product and sum can be partitioned as shown in Figure 15.

Rather than increase the size of the LUT by a growth factor of 16, four LUTs the same size as the one used in the example of Figure 13 are used. The additional three adders, each occupying approximately the same space as one 4-bit LUT, result in a growth factor of between 6 and 7. In a SDA-based design with multiple four-input LUTs, each are configured in the same manner as was discussed with the single four-input LUT.
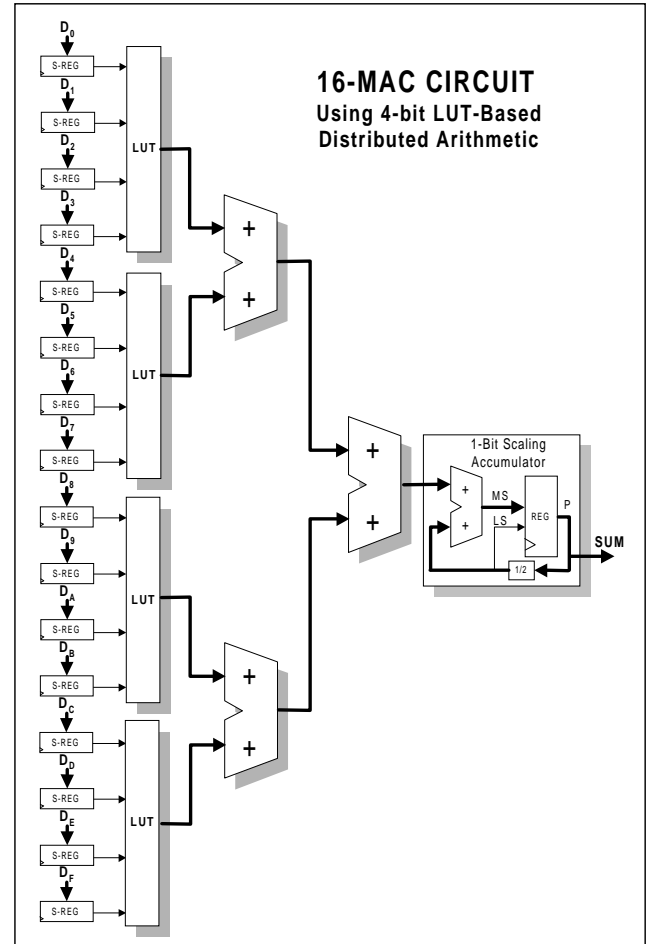

Figure 15.   LUT-Based Serial Distributed Arithmetic for a four-product MAC.

The only difference between a MAC circuit and a FIR filter design is how the input data is handled [see Figure 16]. The MAC has a series of parallel-to-serial converters, while the FIR filter has one long bit-wide shift register that taps into each word. Only the first word is parallel loaded. The first shift register operates by parallel-loading the "new" data sample into a register-based shift register. The shift register then serially shifts out all the data-bits, 1-bit at a time. The output bit from the shift register addresses 1-bit of a corresponding LUT, until the most significant bit of n-bits has been clocked out. When the shifting is complete, the first data shift register is empty and is ready to be parallel loaded with the next word and the process repeats.

The output bit from the first shift register (the first filter Tap) is the input to a serial-in and serial-out shift register (the second filter Tap). Each additional Tap would also use a serial-in and serial-out shift register with its input fed serially from the previous cascaded Tap. Note that

the data flow can be changed to adapt to the input characteristics of any function.

With the exception of the first parallel-in and serial-out shift register, the shift registers could be constructed using registers or more efficiently implemented in a Synchronous RAM-based shift register. This is a result of needing access to only one bit of each data sample (Tap) per clock cycle. The Synchronous RAM-based shift register offers 16 times the density of a register-based shift register (each CLB can be configured as two Registers, two 16x1-bit Synchronous RAMs or one 32x1-bit Synchronous RAM for data storage).

The data in the remaining RAM-based shift registers are positioned to be multiplied by the appropriate coefficient by addressing its corresponding LUT during the next cycle. The oldest sample's (the last Tap) data-bit is lost at the end of each process.
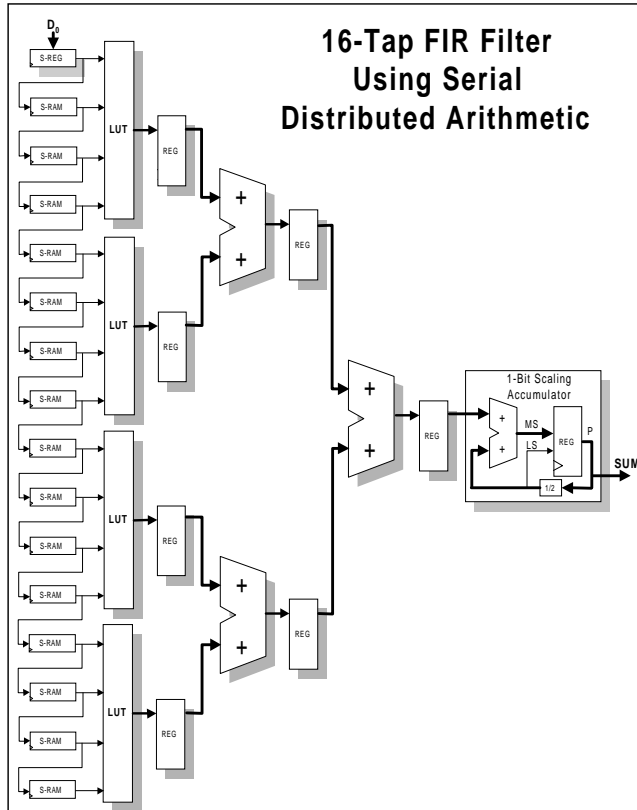

Figure 16. **16-Tap FIR Filter using LUT-Based Serial Distributed Arithmetic.**

The outputs from each LUT and ADDER can be registered in the same CLB that holds the LUT or ADDER at no extra cost. By adding pipeline registers the design can be clocked at a higher rate. For the 16-Tap FIR filter example, the pipelining registers result in a continuous output data stream with a latency of 4-sample cycles, as shown in Figure 17.
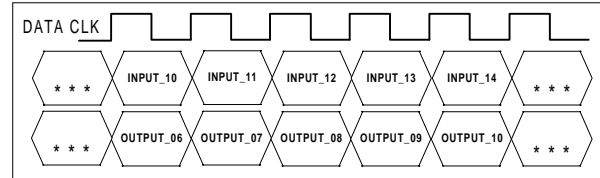

Figure 17. **Pipelined input/output timing diagram for a SDA 16-Tap FIR Filter.**

If the filter is symmetrical it is possible to further reduce the resources required for the design. In a symmetrical filter the number of LUTs (or MACs) can be reduced by a factor of two. This is done by adding the input data with a common coefficient prior to the multiplication. This is done with serial adders. This also reduces the number of adder stages in the design, as shown in Figure 19.
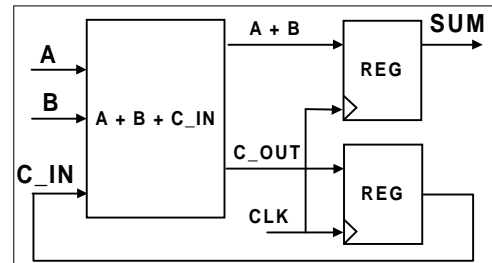

Figure 18. **Serial Adder input/output diagram.**

When serial adders are used to reduce the number of MACs in a Serial Distributed Arithmetic algorithm, an extra clock cycle is required to flush the Carry_Out from the addition of the two most significant bits of the input data [see Figure 18]. The processing clock rate can be calculated by dividing the Sample Rate by the number of sample bits plus one.

A filter design implemented in an FPGA with SDA gives a significant amount of performance in a modest number of CLBs (e.g. 4.25-CLBs at 7.7 nsec, per Tap). SDA uses the smallest number of CLBs while processing all data samples (Taps) in parallel.
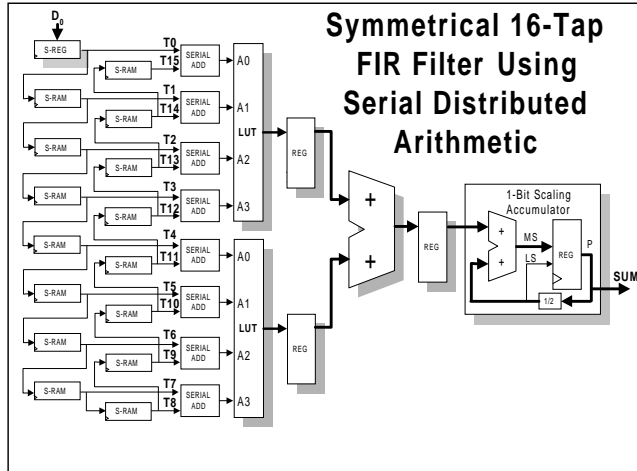
Figure 19.   Symmetrical 16-Tap FIR Filter using LUT-Based Serial Distributed Arithmetic.

A "Serial-Sequential Distributed Arithmetic" (SSDA) technique could be used at lower data rates to conserve more CLBs. This paper does not include a formal discussion on Serial-Sequential Distributed Arithmetic. The process is performed bit-serially, word-sequentially (rather than bit-serially, word-parallel, compared with SDA). This technique uses an internal synchronous Dual-Ported RAM or FIFO to time multiplex each of the data samples, bits serially, through the logic. The performance for Serial-Sequential Distributed Arithmetic is rated at the maximum data rate for a given number of data-bits with SDA, divided by the number of Taps. This technique is typically useful for data sample rates below 1 MHz.

Parallel Distributed Arithmetic:

"Parallel Distributed Arithmetic" (PDA) is used to increase the overall performance of Serial Distributed Arithmetic. With PDA, the number of bits being processed during each clock cycle is increased. Note that increasing the number of bits sampled has a significant effect on the number of CLBs used for the design. Therefore, the number of parallel bits sampled should be increased only to meet the required performance.

Increasing the number of bits processed from 1-bit, in the case of SDA, to a 2-bit PDA results in half the number of processing clock cycles [see Figure 20]. Hence, 2-Bit PDA results in twice the throughput. With 2-bit PDA, the serial shift registers, referenced in the discussion on SDA, are each replaced with two similar 1-bit shift registers at half the bit depth. The two parallel shift registers are spilt, such that one stores the even-bits and the other stores the odd-bits. The 2-bit parallel data samples require twice the number of LUTs. There is also

the addition of a 1-bit scaling adder, required to add the two partial sums which results from each of the two parallel sample bits. The scaling accumulator's input bus is expanded to accommodate the larger partial sum and the final scaling accumulator is changed from a 1- to a 2-bit shift (1/4) for scaling. These changes essentially double the resources required compared to that of the SDA design.
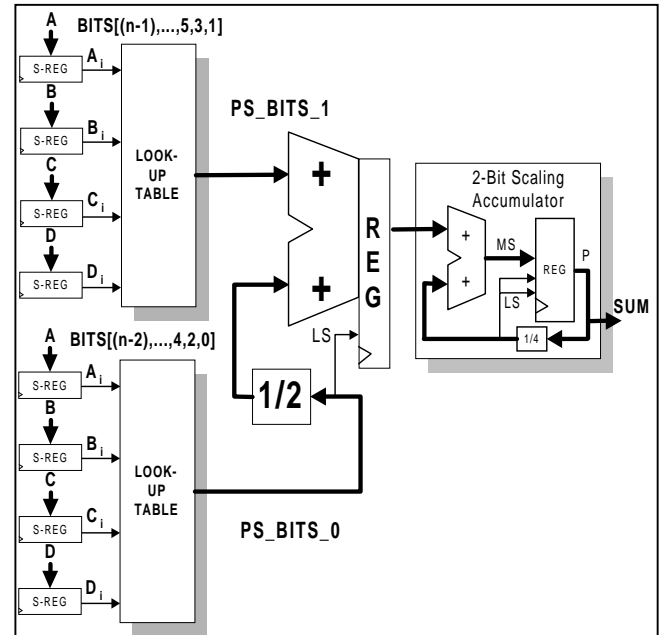

Figure 20.   LUT-Based 2-Bit Parallel Distributed Arithmetic for a four-product MAC.

The performance for the 16-Tap FIR filter example, implemented with a 2-Bit PDA algorithm, resulted with a sample rate of 16 MHz, in 130 CLBs [see Figure 21].

A filter design implemented in an FPGA with 2-Bit PDA results in twice the performance and about twice the number of CLBs (e.g. 8.2-CLBs at 3.9 nsec, per Tap) compared to the same function with SDA. 2-Bit PDA uses a less modest number of CLBs then that of SDA, while still processing all data samples (Taps) in parallel, at twice the SDA data sample rate.

The number of bits being processed during each clock cycle can be increased until an n-Bit PDA implementation is reached, for n-bit data samples. When the design is an n-Bit PDA, the sample data rate is at a maximum. For an XC4000E-3 device this will enable a single chip solution with a sustained data sample rate between 50 and 70 MHz.
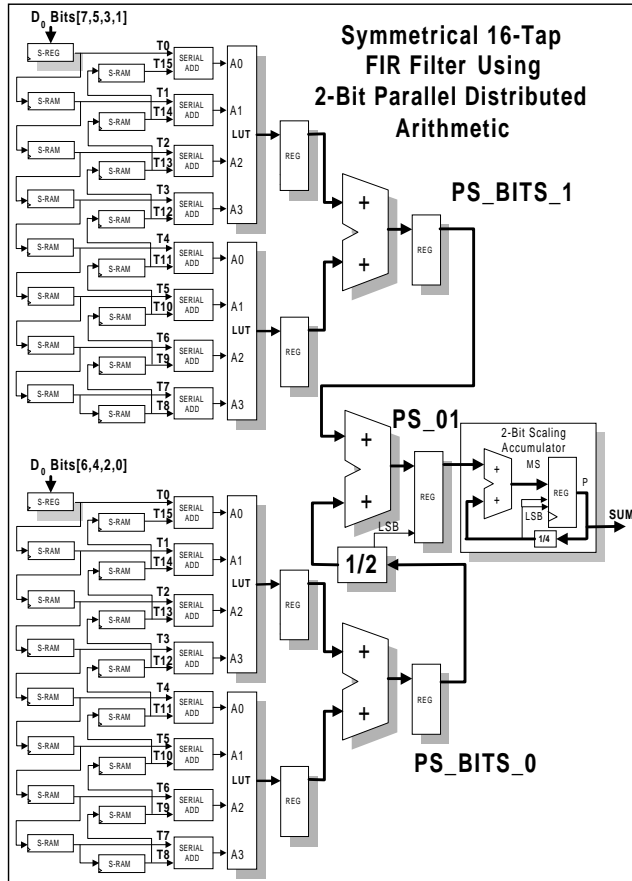
Figure 21.   Symmetrical 16-Tap FIR Filter using 2-Bit Parallel Distributed Arithmetic.

Figure 22.   16-Tap FIR Filter using LUT-Based Parallel Distributed Arithmetic.

With PDA, each additional parallel bit requires an additional level of scaling (by powers of 2) and summation for each partial product pair of bits [see Figure 22].

The LUTs for SDA and PDA can always be the same for any given 4-MAC block, regardless of the number of bits in the sample data. This is true for PDA only if common bit-weighted sample inputs are used to address the LUT, as shown in Figure 22.

The performance for the symmetrical 16-Tap FIR filter example, implemented with an 8-Bit PDA algorithm, resulted in a sample rate of 58 MHz, in 270 CLBs. A non symmetrical 16-Tap FIR filter, as shown in Figure 22, resulted in a sample rate of 55 MHz, in 400 CLBs.
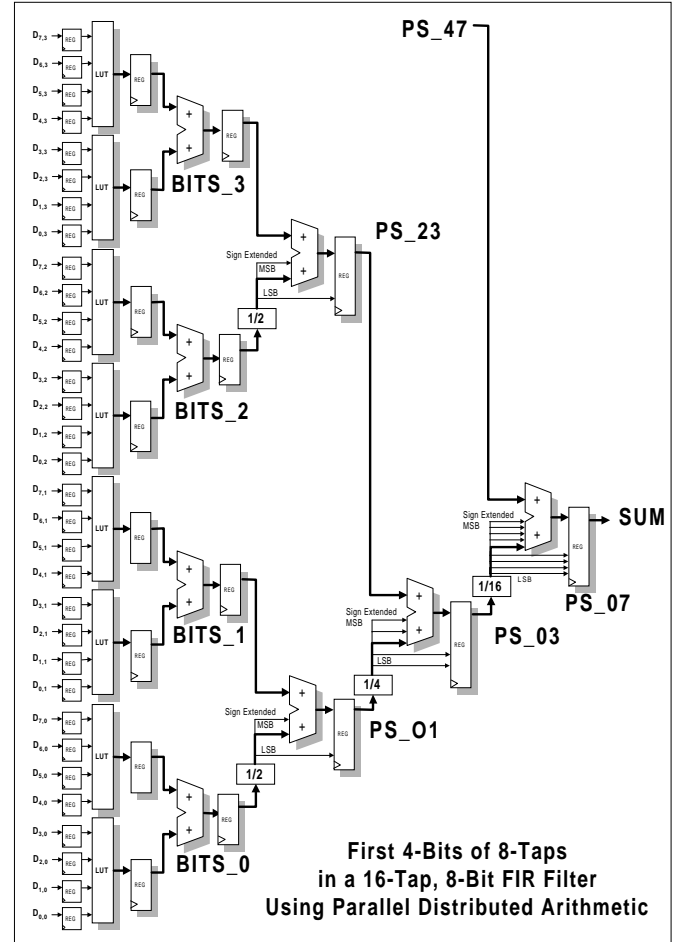
The filter design example implemented in an FPGA with 8-Bit PDA resulted in more than seven times the performance and about four and a half times the number of CLBs (e.g. 19-CLBs at 1.1 nsec, per Tap) compared to the same function with SDA. Full PDA uses a much larger number of CLBs then that of SDA, while still processing all data samples (Taps) in parallel, at the data sample rate.

Other Resources:

DSP Applications Group

For more assistance in FPGA-Based DSP related applications, information or for help on implementing an algorithm in programmable logic, contact the Xilinx DSP Applications Group. Contact them via E-mail at **dsp@xilinx.com** or FAX: **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. "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices," Proceedings of the DSP^X 1995 Technical Proceedings pp. 595-604, 12JAN95.

[2]   Goslin, G. R. & Newgard, B. "16-Tap, 8-Bit FIR Filter Application Guide," Xilinx Publications, 21NOV94.

[3]   "The programmable Logic Data Book," Xilinx, Inc. Second Edition, 1994

[4]   "XC4000E Data Sheet," Xilinx, Inc. Latest Version.

[5]   Strauss, W. "DSP Strategies for the 90's: The Mid-Decade Outlook," Forward Concepts, JAN95.

[6]   "XC6200 Data Sheet," Xilinx, Inc. Latest Version.

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

NOTES: