HIGH-PERFORMANCE DSP CAPABILITY WITHIN AN OPTIMIZED LOW-COST FPGA ARCHITECTURE

A Lattice Semiconductor White Paper

June 2004

Lattice Semiconductor
5555 Northeast Moore Ct.
Hillsboro, Oregon 97124 USA
Telephone: (503) 268-8000
www.latticesemi.com

Introduction

The applications of Digital Signal Processing (DSP) continue to expand, driven by trends such as the increased use of video and still images and the demand for increasingly reconfigurable systems such as Software Defined Radio (SDR). Many of these applications combine the need for significant DSP processing with cost sensitivity, creating demand for high-performance, low-cost DSP solutions.

General-purpose DSP chips and FPGAs are two common methods of implementing DSP functions. Each approach has advantages, and the optimum implementation method will vary depending upon application requirements. This white paper provides an overview of common DSP functions and then explores the differences between the general purpose DSPs and FPGAs. This is followed by a description of the LatticeECP™-DSP (EConomy Plus Digital Signal Processing) architecture and a comparison of the LatticeECP-DSP to existing FPGA solutions.

Typical Functions

While a vast array of digital signal processing functions are implemented by designers, Finite Impulse Response (FIR) filters, Infinite Impulse Response (IIR) filters, Fast Fourier Transforms (FFTs) and mixers are common to many applications. Each of these functions requires a combination of multiply elements along with addition, subtraction and accumulation. This section provides a brief overview of the algorithms used to implement these functions.

Finite Impulse Response (FIR) Filters

The finite impulse response filter stores a series of n data elements, each delayed by an additional cycle. These data elements are commonly referred to as taps. Each tap is multiplied by a coefficient and the results summed to produce the output. Some implementations perform all the multiplications in parallel. More generally, the implementation is broken down into N stages, with an accumulator passing the partial result from one stage to the next. This implementation trades speed for functional

resources, taking N computation stages and requiring n/N multipliers. Depending upon whether the coefficients are static or dynamic, and the design of the coefficient values, there are a number of other design optimizations commonly used that are beyond the scope of this paper. Figure 1 shows the implementation of a typical FIR filter.

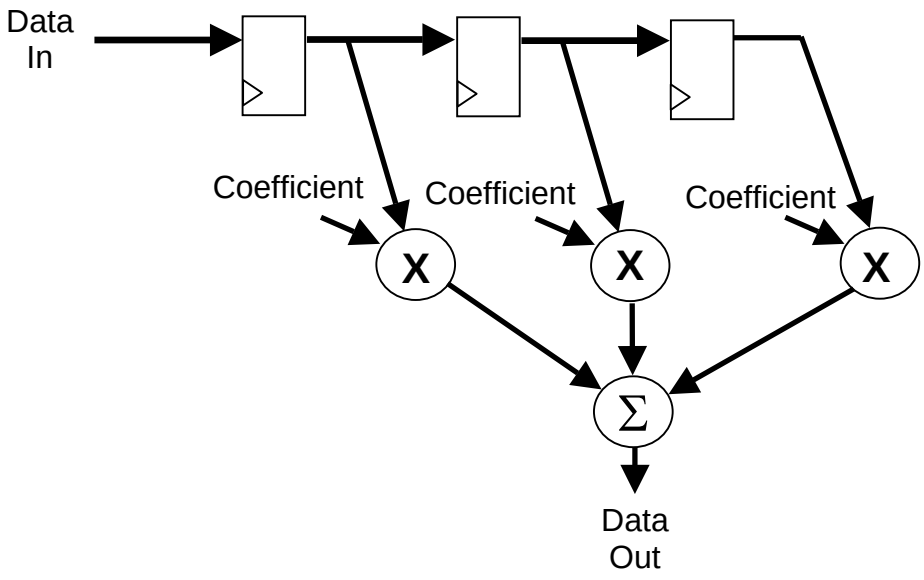

Figure 1 -- Typical FIR Filter

Fast Fourier Transform (FFT) Functions

Fast Fourier Transforms are used for a variety of applications, ranging from image compression to determining the spectral content of a data sample. There are a variety of methods for implementing the Fast Fourier Transform. Probably the most common method is to use Cooley-Tukey decimation in time approach, which breaks the FFT down into a number of smaller FFTs. The simplest implementation uses an element commonly referred to as the Radix-2 butterfly, through which the input data must be passed multiple times. Figure 2 shows the Radix-2 Butterfly. The calculation is conceptually simple, as shown on the left of the diagram. However, as all the multiplies and additions are done with complex numbers, the actual number of multiplies and additions required is somewhat more challenging, as shown on the right side of the diagram.

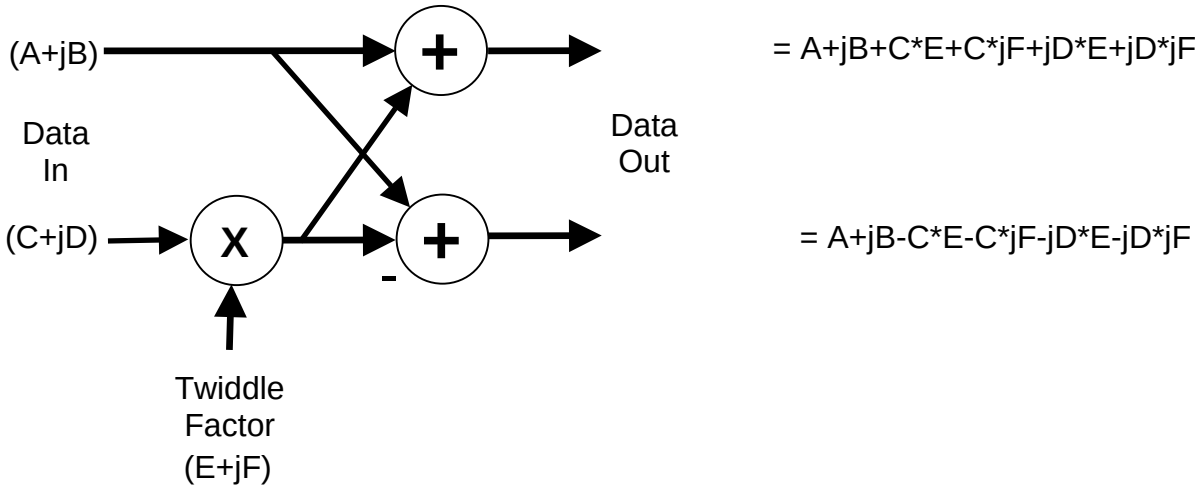

Figure 2 – Radix-2 Butterfly Commonly Used For Implementing FFTs

Infinite Impulse Response (IIR) Filters

The Infinite Impulse Response (IIR) filter is similar to the FIR filter, except that feedback paths are introduced. These feedback paths make the design and analysis of IIR filters more complex than FIR filters. However, the IIR approach can provide a more powerful filter for the same silicon area. Although there are several IIR architectures, one common approach is to build IIR filters out of second order bi-quads, as shown in figure 3.

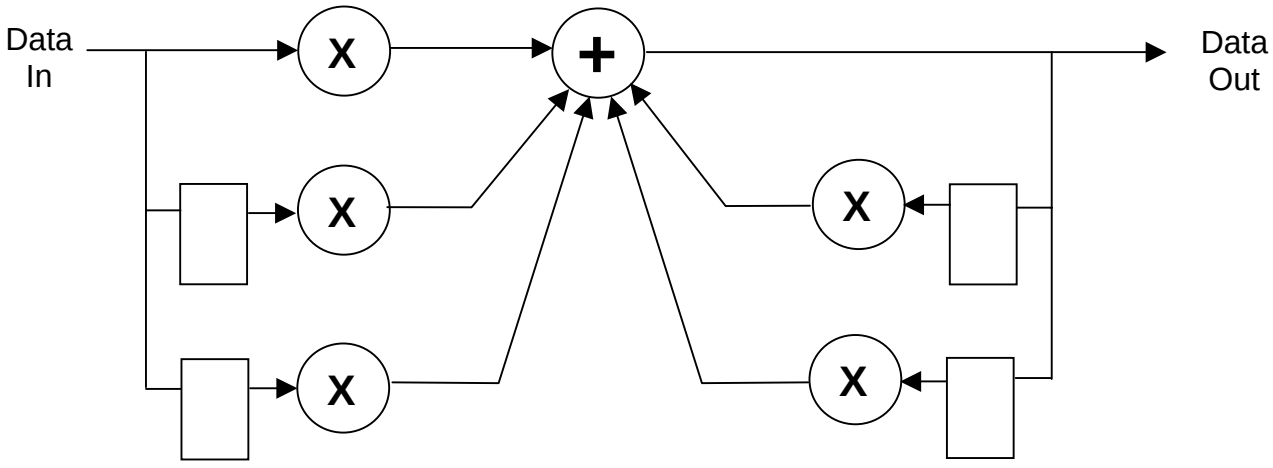

Figure 3 – IIR Second Order Bi-quad

Mixer Functions

Many applications use mixers to shift the frequency of a signal. While, conceptually, just a single multiplier could be used, in digital applications there are a number of advantages to representing the numbers in a complex form. Most typically this is done by representing signals as I and Q components. Figure 4 shows a mixer that would be used in digital up-conversion.

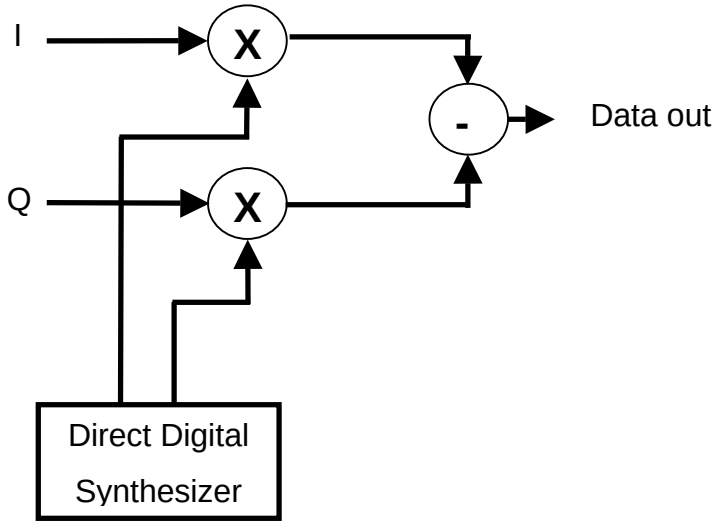

Figure 4 – Typical Up Converter Mixer using Complex Arithmetic

General Purpose DSP Solutions Versus FPGA Implementations

As illustrated in the description of common functions, multipliers, followed by addition, subtraction or accumulation are at the heart of most DSP applications. General-purpose DSP chips combine efficient implementations of these functions with a general-purpose microprocessor. The number of multipliers is generally in the range of one to four, and the microprocessor will sequence data to pass it through the multiply and other functions storing intermediate results in memory or accumulators. Performance is increased primarily by increasing the clock speed used for multiplication. Typical clock

speeds run from tens of MHz to 1GHz. Performance, as measured by Millions of Multiply Accumulates (MMAC) per second, typically ranges from 10 to 4000. Functions requiring higher performance have to be split across multiple DSP engines. The price of these chips ranges from a few dollars at the bottom end of the performance range to hundreds of dollars at the high end. The key advantage of this approach is the ability to directly implement algorithms written in a high-level programming language such as C.

DSP oriented FPGAs provide the ability to implement many functions in parallel on one chip. General-purpose routing, logic and memory resources are used to interconnect the functions, perform additional functions, sequence and, as necessary, store data. Some basic devices provide multiplier only support, requiring users to construct all other functions in logic. More sophisticated devices provide addition, subtraction and accumulator functions as part of their set of DSP building blocks. FPGAs typically have tens of multiplier elements and can operate at clock speeds of hundreds of MHz. For example, the LatticeECP-DSP 20 FPGA has 28 18x18 multipliers that can run at speeds up to 250MHz, delivering performance up to 7,000 MMAC per second. Table 1 compares the FPGA and general-purpose approach.

Device	Clock Speed	Number of Multipliers	MMAC/s	1K Unit Cost*	Cost per MMAC/s
TI DSP	1GHz	4	4000	$256	$0.064
TI DSP	300MHz	4	1200	$40	$0.033
ECP-DSP20	250MHz	28	7000	$59	$0.008

* Approximate 1K pricing through NA distributors

Table 1 – Comparison of General-Purpose DSP and FPGA approaches

LatticeECP-DSP Architecture

The LatticeECP-DSP devices consist of a low-cost FPGA fabric coupled with between four and ten sysDSP™ blocks. Figure 5 shows the overall block diagram of the ECP device[i]. The sysDSP block in the LatticeECP family supports four functional elements in three data path widths: 9, 18 and 36. The user selects a function element for a DSP block and then selects the width and type (signed/unsigned) of its operands. The operands in the sysDSP Blocks can be either signed or unsigned, but not mixed within a

function element. Similarly, the operand widths cannot be mixed within a block. The resources in each sysDSP block can be configured to support the following four elements:

- MULT (Multiply, Figure 6)
- MAC (Multiply Accumulate, Figure 7)
- MULTADD (Multiply Addition/Subtraction, Figure 8)
- MULTADDSUM (Multiply Addition/Subtraction Summation, 9)

The number of elements available in each block depends upon the width selected from the three available options: x9, x18, and x36. A number of these elements are concatenated for highly parallel implementations of DSP functions. Table 2 shows the capabilities of the block.

Width of Multiply	X9	X18	X36
MULT	8	4	1
MAC	4	2	--
MULTADD	4	2	--
MULTADDSUM	2	1	--

Table 2 – Maximum Number of Elements in a sysDSP Block

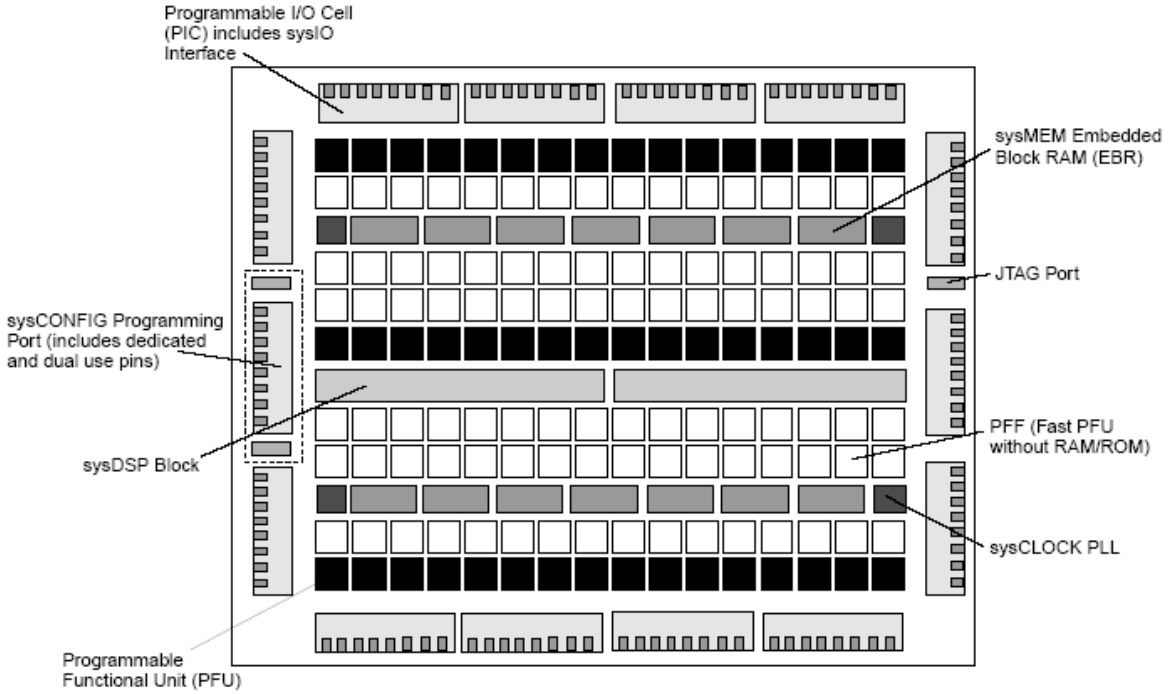

Figure 5 – LatticeECP-DSP Block Diagram

The sysDSP block has built-in optional pipelining at the input, intermediate and output stages. In addition, inputs can be loaded in parallel or shifted across the array as necessary. Options are also provided for dynamically switching between signed and unsigned arithmetic and subtraction and addition.

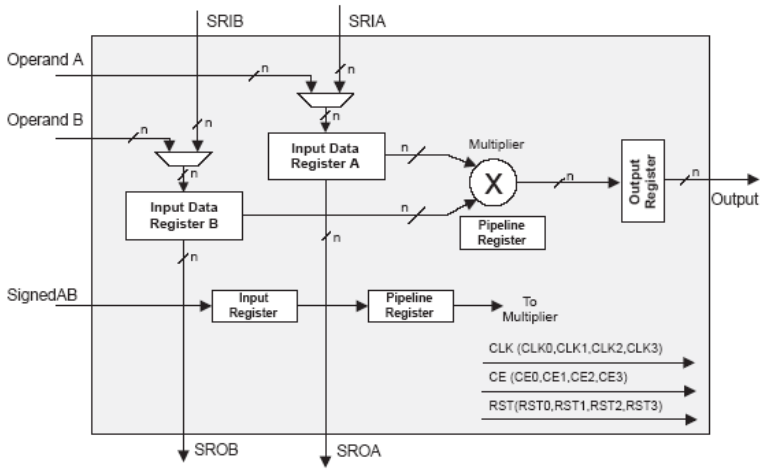

Figure 6– MULT (Multiplier) Element

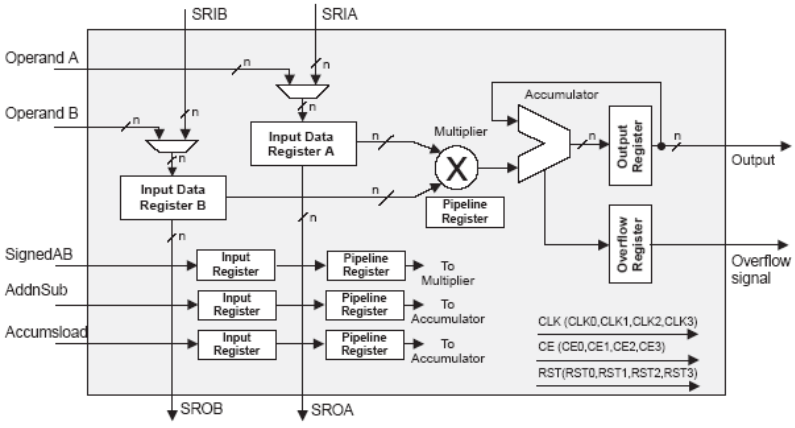

Figure 7 – MAC (Multiply Accumulate) Element

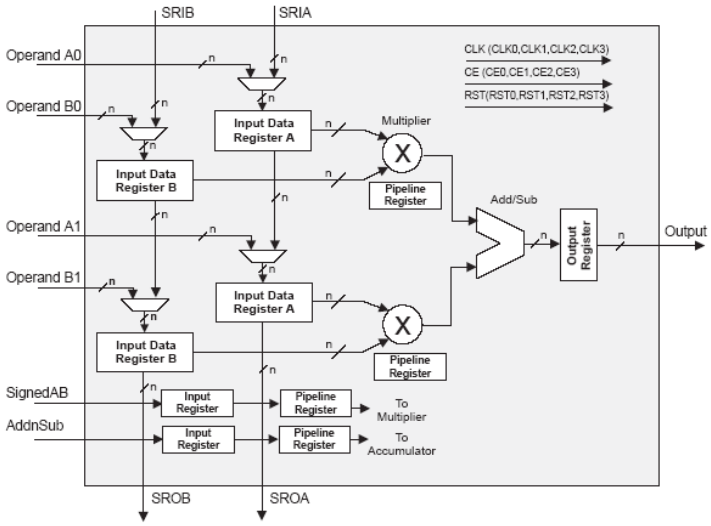

Figure 8 – MULTADD (Multiplier Addition/Subtraction) Element

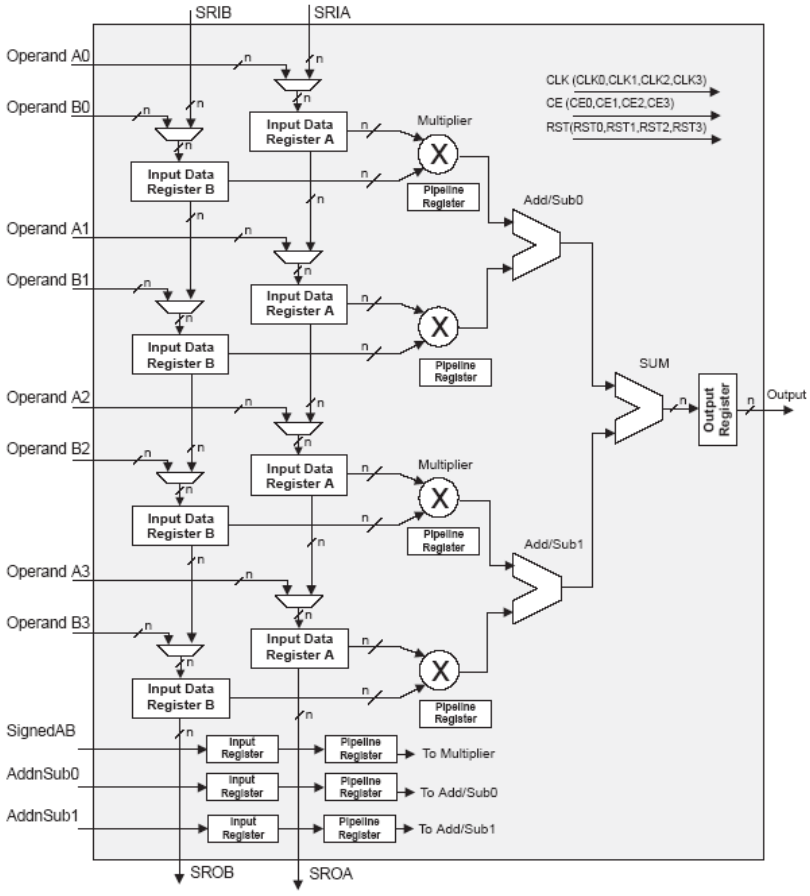

Figure 9 – MULTADDSUM (Multiplier Addition/Subtraction Summation) Element

Performance and Device Utilization Improvements

The availability of pipelining registers, summation, subtraction and accumulation within the sysDSP blocks increases their utility. As illustrated, in typical functions it is very common to need to combine multiplication with addition, summation, or accumulation. Pipelining registers, while conceptually simple, rapidly consume significant resources when implemented on wide data paths. The sysDSP blocks' ability to implement these functions results in lower consumption of general-purpose FPGA resources and higher performance. Both of these factors translate directly into lower costs, as in many cases they allow designers to select smaller devices with lower speed grades.

LatticeECP-DSP Design Flow

Lattice provides designers with four simple methods to access the capabilities of the sysDSP Block:

- The **Module/IP Manager** is a graphical interface provided in the ispLEVER® tools that allows the rapid creation of modules implementing DSP elements. These modules can then be used in HDL designs as appropriate.
- The coding of certain functions into a design's HDL and allowing the synthesis tools to **Inference** the use of a DSP block.
- The implementation of designs in **MathWork's Simulink** tool using a Lattice Block set. The ispLeverDSP portion of the ispLEVER tools will then convert these blocks into HDL as appropriate.
- **Instantiation** of DSP primitives directly in the source code.

The method chosen for any design will depend upon the DSP algorithm design methodology and the degree of control desired over the physical implementation. Figure 10 illustrates the specification of a MULT element using the module manager. Figure 11 shows the use of Lattice block sets in MathWork's Simulink tool.

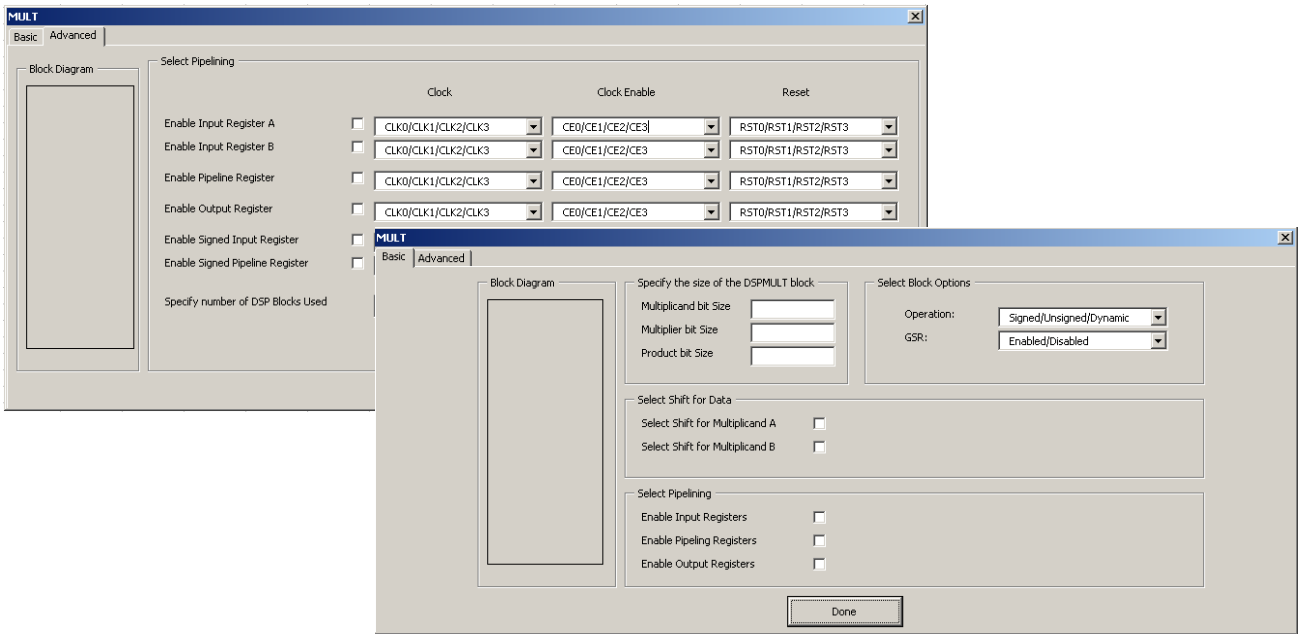

Figure 10 – Configuring A Multiplier Element in the Module Manager

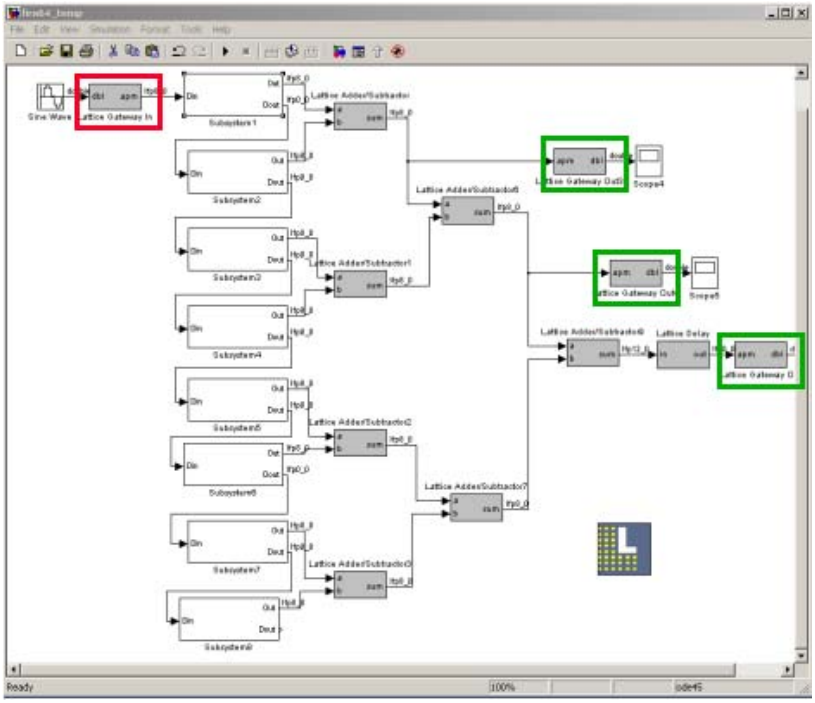

Figure 11 – Using the Lattice Block Set In MathWork's Simulink Tool

Low-Cost FPGA Implementations

With the introduction of the LatticeECP/EC devices, users can now choose among three current generation, low-cost FPGAs: the Spartan III devices from Xilinx, Altera's Cyclone family and the LatticeECP/EC devices.

Altera's Cyclone FPGAs contain no DSP oriented element, making it challenging to implement large DSP functions in these devices without consuming a significant number of internal resources. Naturally, achieving high-performance with these implementations is equally challenging. The Xilinx Spartan III FPGA family does provide some basic multiplier capability. While this is certainly preferable to having no DSP capability at all, significant resources must still be consumed to implement the adders, subtractors, accumulators and pipeline registers found in typical designs.

To measure the effect of providing these resources, Lattice benchmarked performance and utilization for a FIR filter and an IIR filter. The FIR used was a 64-tap filter with 18-bit wide data. The IIR filter used was 4th order arranged as two bi-quads and an 18-bit data path. Figure 12 shows the results for both the Spartan III and LatticeECP-DSP devices.

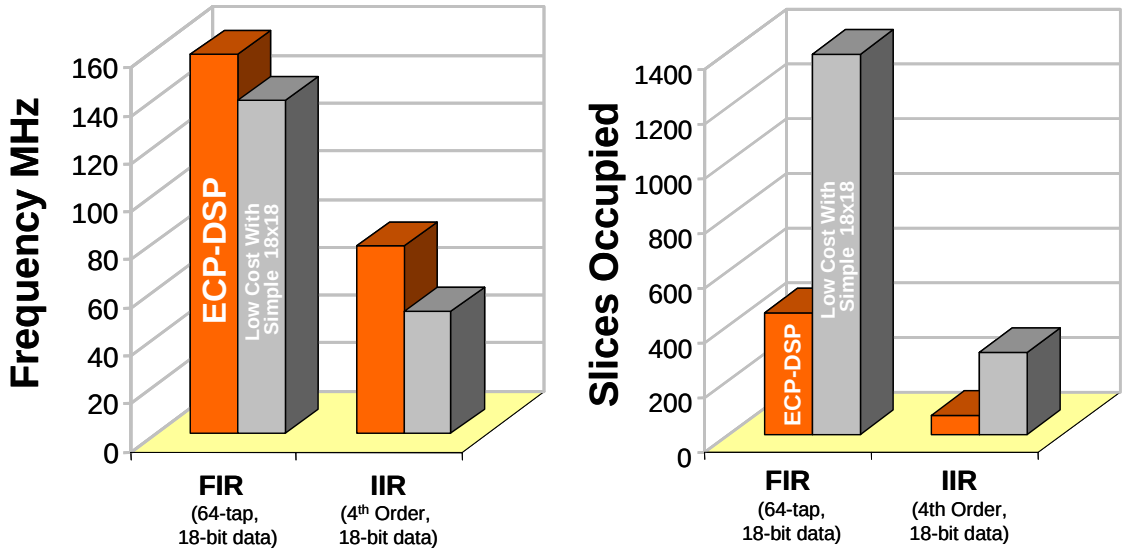

Figure 12 – FIR and IIR implementations in LatticeECP-DSP and Spartan III Devices

Summary

The use of DSP techniques will continue to grow at the expense of analog implementations. An analysis of the functions typically used in DSP applications indicates that a combination of multiplier, addition, subtraction and accumulation elements is required. The LatticeECP devices provide a sophisticated DSP block combined with a low-cost FPGA fabric. Through the implementation of addition, subtraction, accumulation and pipelining within the sysDSP block, performance and LUT utilization are considerably higher than those of alternative low-cost FPGA solutions that provide only basic multiplier capabilities. The speed and utilization advantages of the sysDSP block help users reduce costs through the selection of smaller and lower speed grade devices.

###

---

i Please see the Lattice Whitepaper “Optimizing FPGAs For High-Volume Applications” for more details on the FPGA fabric used for the LatticeECP-DSP devices.