# An Assessment of the Suitability of FPGA-Based Systems for use in Digital Signal Processing* **

Russell J. Petersen and Brad L. Hutchings

Brigham Young University, Dept. of Electrical and Computer Engineering, 459 CB, Provo UT 84602, USA

**Abstract.** FPGAs have been proposed as high-performance alternatives to DSP processors. This paper *quantitatively* compares FPGA performance against DSP processors and ASICs using actual applications and existing CAD tools and devices. Performance measures were based on actual multiplier performance with FPGAs, DSP processors and ASICs. This study demonstrates that FPGAs can provide an order of magnitude better performance than DSP processors and can in many cases approach or exceed ASIC levels of performance.

## 1 Introduction

To meet the intensive computation and I/O demands imposed by DSP systems many custom digital hardware systems utilizing ASICs have been designed and built. Custom hardware solutions have been necessary due to the low performance of other approaches such as microprocessor-based systems, but have the disadvantage of inflexibility and a high cost of development. The DSP *processor* attempts to overcome the inflexibility and development costs of custom hardware. The DSP processor provides flexibility through software instruction decoding and execution while providing high performance arithmetic components such as fast array multipliers and multiple memory banks to increase data throughput. The FPGA has also recently generated interest for use in implementing digital signal processing systems due to its ability to implement custom hardware solutions while still maintaining flexibility through device reprogramming [2]. Using the FPGA it is hoped that a significant performance improvement can be obtained over the DSP processor without sacrificing system flexibility. This paper is an attempt to quantify the ability of the FPGA to provide an acceptable performance improvement over the DSP processor in the area of digital signal processing.

* To be published in 5th International Workshop on Field-Programmable Logic and Applications, Oxford, England, Aug. 1995.

** This work was supported by ARPA/CSTO under contract number DABT63-94-C-0085 under a subcontract to National Semiconductor.

# 2 Multiplication and digital signal processing

A core operation in digital signal processing algorithms is multiplication. Often, the computational performance of a DSP system is limited by its multiplication performance, hence the multiplication rate of the system must be maximized. Custom hardware systems based on ASICs and DSP processors maximize multiplication performance by using fast parallel-array multipliers either singly or in parallel. FPGAs also have the ability to implement multipliers singly or in parallel according to the needs of the application. Thus, in order to understand the performance of the FPGA relative to the ASIC and the DSP processor a comparison of FPGA multiplication alternatives and their performance relative to custom multiplier solutions is needed. This section presents the basic alternatives for multiplier implementations and their performance when implemented on FPGAs.

## 2.1 Multiplier architecture alternatives

When implementing multipliers in hardware two basic alternatives are available. The multiplier can be implemented as a fully parallel-array multiplier or as a fully bit-serial multiplier as shown in Figure 1. The advantage of the fully parallel approach is that all of the product bits are produced at once which generally results in a faster multiplication rate. The multiplication rate for a parallel multiplier is just the delay through the combinational logic. However, parallel multipliers also require a large amount of area to implement. Bit-serial multipliers on the other hand generally require only $\frac{1}{N}$th the area of an equivalent parallel multiplier but take 2N bit times to compute the entire product (N is the number of bits of multiplier precision). This often leads one to believe that the bit-serial approach is thus 2N times slower than an equivalent parallel multiplier but this is not true. The bit-times (clock cycles for synchronous bit-serial multipliers) are very short in duration due to the reduced size and hence propagation paths of the multiplier. This results in a bit-serial multiplier achieving about $\frac{1}{2}$ the multiplication rate of an equivalent parallel multiplier on average, even exceeding the performance of the parallel multiplier in some cases.

## 2.2 FPGA multiplication results

Table 1 lists the performance of several multipliers implemented on three different FPGAs. The FPGAs used were a Xilinx 4010, an Altera Flex8000 81188, and a National Semiconductor CLAy31. The first two FPGAs can be characterized as medium-grained architectures and are approximately equivalent in logic-density while the last FPGA is a fine-grained architecture utilizing smaller but more numerous cells. The multiplication rate of each multiplier is listed in MHz as well as the percentage of the FPGA required to implement the multi-

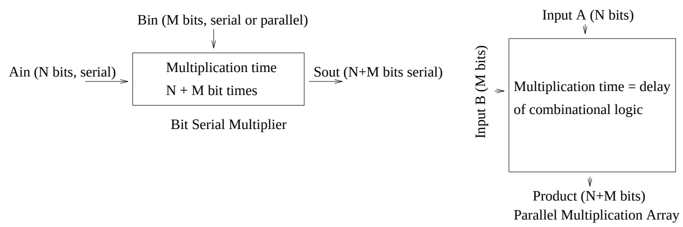


**Fig. 1.** Block diagrams of basic multiplier alternatives

plier. The bit-serial multipliers have listed both their clock rate (bit-rate) and their effective multiplication rate (clock rate/2N).

## 2.3 Multiplier table contents

The majority of the multipliers in this study used common architectures such as the Baugh-Wooley two's complement parallel-array multiplier [5] and pipelined versions of the bit-serial multiplier [6] shown in Figure 1. In addition, several custom parallel multipliers were built that take advantage of the special features available on the Altera and Xilinx FPGAs. These are intended to represent near the absolute maximum possible multiplier performance that can be achieved with these current FPGAs. These specific customizations will be discussed below.

Several of the multipliers listed in the tables have the label synthesis attached. This label indicates that the multipliers were created by synthesizing simple high-level hardware language (VHDL) design statements (`z <= a * b`). These multipliers were included so as to allow a comparison between hand-placed multipliers using schematics and high-level language designed multipliers. The table results show that the synthesized multipliers performed very favorably as shown in the Xilinx 4010 parallel multiplier table section. The 8 and 16-bit unsigned and signed array multipliers listed first were designed with schematics and were hand placed onto the FPGA. However, their performance was nearly identical in terms of both speed and area required to the multipliers synthesized from VHDL.

### 2.3.1 Fast carry-logic based parallel multipliers

The Altera 81188 based multipliers labeled *fast adder* refer to the use of the fast carry-logic available on the Altera FPGAs to make fast ripple-carry adders. These adders are then used to build fast multipliers by using the adders to add the successive partial product rows. This technique results in multipliers that are approximately twice as fast on the FPGAs as those not implemented with

**Table 1.** FPGA Multiplier Performance Results

| Type of Multiplier | # CLB/LC's | % of FPGA | Mult. Speed |
|---|---|---|---|
| **Altera 81188 Parallel Multipliers** | | | |
| 8-bit unsigned fast-adder | 133 | 13 | 14.8 MHz |
| 8-bit signed fast-adder | 150 | 14 | 12.8 MHz |
| 8-bit unsigned synthesis | 129 | 12 | 7 MHz |
| 8-bit signed synthesis | 135 | 13 | 6.84 MHz |
| 8-bit signed complex synthesis | 584 | 57 | 5.86 MHz |
| 16-bit unsigned fast-adder | 645 | 63 | 3.34 MHz |
| 16-bit unsigned synthesis | 519 | 51 | 3.66 MHz |
| 16-bit signed synthesis | 535 | 53 | 3.4 MHz |
| **Altera 81188 Bit-Serial Multipliers** | | | |
| 8-bit unsigned | 29 | 3 | 84.03/5.25 MHz |
| 8-bit signed | 91 | 9 | 69/4.6 MHz |
| 16-bit unsigned | 61 | 7 | 68.49/2.14 MHz |
| 16-bit signed | 186 | 18 | 64/2 MHz |
| **National Semiconductor CLAy Parallel Multipliers** | | | |
| 8-bit unsigned | 329 | 11 | 7.9 MHz |
| 8-bit signed | 338 | 11 | 7.2 MHz |
| 16-bit unsigned | 1425 | 45 | 3.6 MHz |
| 16-bit signed | 1446 | 46 | 3.53 MHz |
| **National Semiconductor CLAy Bit-Serial Multipliers** | | | |
| 8-bit unsigned | 48 | 1.5 | 32.2/2.01 MHz |
| 8-bit signed | 48 | 1.5 | 32.2/2.01 MHz |
| 16-bit unsigned | 96 | 3 | 29.2/.91 MHz |
| 16-bit signed | 96 | 3 | 29.2/.91 MHz |
| **Xilinx 4010 Parallel Multipliers** | | | |
| 8-bit unsigned | 64 | 16 | 8.54 MHz |
| 16-bit signed | 259 | 65 | 4.35 MHz |
| 8-bit unsigned synthesis | 61 | 15 | 9 MHz |
| 8-bit signed synthesis | 61 | 15 | 8 Mhz |
| 8-bit signed complex synthesis | 266 | 66 | 7.3 MHz |
| 16-bit unsigned synthesis | 242 | 60 | 3.8 MHz |
| 16-bit signed synthesis | 250 | 63 | 3.7 MHz |
| **Xilinx 4010 Bit-Serial Multipliers** | | | |
| 8-bit unsigned | 17 | 4 | 73.1/4.6 MHz |
| 8-bit signed | 32 | 8 | 52/3.3 MHz |
| 16-bit unsigned | 33 | 8 | 62/1.9 MHz |
| 16-bit signed | 64 | 16 | 50/1.6 MHz |
| **Xilinx 4010 Parallel Constant Multipliers** | | | |
| 8-bit unsigned ROM | 22 | 5.5 | 21.7 MHz |
| 16-bit unsigned ROM | 84 | 21 | 11.36 MHz |
| 8-bit unsigned RAM | 39 | 9.75 | 17.86 MHz |
| 16-bit unsigned RAM | 117 | 29.3 | 10.4 MHz |

special logic. The disadvantage of this approach is the resulting difficulty that arises with the placement of the multiplier onto the FPGA. The FPGA router is only able to place three of the unsigned 8-bit multipliers on a 81188 FPGA even though they only utilize 13% of the total FPGA resources each.

### 2.3.2 Constant multipliers and distributed arithmetic

The use of constants (constant multiplicand ) in multiplication can significantly reduce the size of a parallel multiplier array. This is because the presence of zeros in the constant can result in the elimination of many partial product terms in the multiplication array. This technique is especially useful in DSP systems since many of the multiplications to be performed can be specified in terms of constant multipliers. For example, with an FIR filter each tap of the filter can be implemented using a multiplier with a constant tap coefficient.

The use of constants in multiplication also makes available another technique that can result in a significant multiplier performance increase. This technique is called the *distributed arithmetic* approach to multiplication and can be implemented by the Xilinx FPGAs due to their ability to provide small blocks of distributed RAM to be used as partial-product lookup tables.

The distributed arithmetic approach to multiplication relies upon the ability to easily precompute all of the possible products of a multiplication when one of the values is held constant. For example, consider an 8x8 bit multiplier implemented with this technique. One, possibility is to break up the 8-bit input word into two nibbles (4 bits) and then use each nibble as the address applied to two separate 12-bit wide, 16-location lookup tables. Two separate 16x12 bit tables are required since each of the nibbles produces 16 possible 12-bit partial products. The partial product outputs of each table are then weighted appropriately and added to produce the product. The method is illustrated in Figure 2. The partial product produced by the high-order nibble of the input word is shifted by 4 bits to the left (a weighting factor of 16) and added to the partial product produced by the lower-order nibble of the input word to produce the 16 bit output [3].

Implementing the 8x8 multiplier on a Xilinx FPGA requires a total of 384 bits of storage along with a 12 bit adder. This results in a minimum of 12 CLBs for the data storage and approximately 12 CLBs for the adder for a total of 24 CLBs. The actual number of CLBs (area) required is dependent upon optimizations that the place-and-route software is able to make and can be seen to be slightly less (22 CLBs) for the ROM-based 8x8 multiplier in Table 1. The difference in size and speed between the RAM and ROM-based versions listed in the table is due to the elimination of the additional inputs on the ROM version and the associated optimizations that the place-and-route software can make. Only unsigned constant Xilinx multipliers are listed in the table but signed versions of the multipliers can also easily be built by sign-extending the partial products and the input to the multiplier.

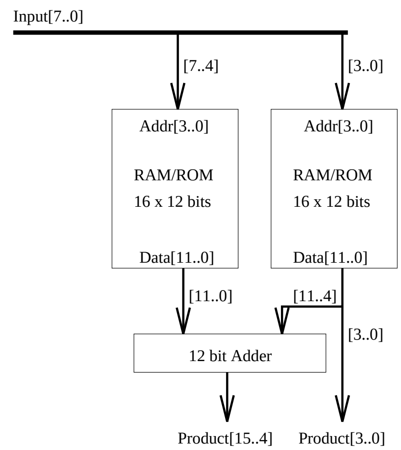


**Fig. 2.** 8-bit constant unsigned multiplier using distributed arithmetic

### 2.4 Comparisons to custom multiplier chips

One possible alternative to implementing multipliers on an FPGA is to use external multiplication chips with the FPGA providing the necessary control. This allows the use of multipliers designed in VLSI that are faster, smaller, and less expensive than equivalent implementations on FPGAs. The table below lists several fixed-point multiplication chips available from various manufacturers along with their performance.

**Table 2.** Custom Multiplier Chip Performance

| Part # | Precision | Mult. Speed |
|---|---|---|
| Logic Devices LMU08/LMU8U | 8x8-16 bit signed/unsigned | 28.6 MHz |
| Logic Devices LMU18 | 16x16-32 bit signed/unsigned | 28.6 MHz |
| Cypress CY7516/517 | 16x16-32 bit signed/unsigned | 26.3 MHz |
| GEC Plessey PDSP16116/A | 16-64 bit signed/unsigned complex | 20 MHz |

Disadvantages of using external multipliers include the on/off chip time required for signals between the FPGA and the multiplier and the high I/O pin requirement when interfacing to a multiplication chip. For example, the 16-bit complex multiplier requires 128 pins just for data transfer. Some of the I/O constraints are eased with the 16-bit multipliers by multiplexing the inputs with the output data word but this also requires extra control and adds latency to the multiplier.

As can be seen from Tables 1 and 2 the FPGA-based parallel multipliers obtain approximately $\frac{1}{4}$ to $\frac{1}{3}$ of the performance of the custom multipliers for the 8-bit versions while the 16-bit multipliers obtain only about $\frac{1}{10}$ the performance of their custom counterparts. The only FPGA-based multipliers that come close to matching the custom multiplier performance are the constant multipliers based on the distributed arithmetic approach.

# 3 Performance comparison of two popular DSP algorithms

Using the previous results for multiplication, rough comparisons can be made between the performance of FPGA-based, DSP processor, and ASIC-based DSP systems. Two popular DSP algorithms that have been chosen for this comparison are a single-dimensional FIR filter and a FFT. Comparisons will be made based on implementations using: FPGAs only, FPGAs combined with external multiplier chips, a single DSP processor, and full custom ASICs. In the comparisons it will be assumed that the multipliers form the limiting path of the system and that an additional 10 ns is required for on/off chip delays between the multiplier and the FPGA when using the external multiplication chips.

**Table 3.** 20-Tap FIR Filter Performance

| System | Precision | # of Chips | Computation Time | Data rate |
|---|---|---|---|---|
| TI TMS320C5X | 16 bit | 1 | 1.0 $\mu$s | 738.9 KHz |
| Altera 81188 U-Bit-Serial | 8 bit | 1 | .190 $\mu$s | 5.3 MHz |
| Altera 81188 U-Bit-Serial | 16 bit | 2 | .477 $\mu$s | 2.1 MHz |
| Altera 81188 S-Bit-Serial | 8 bit | 3 | .227 $\mu$s | 4.4 MHz |
| Altera 81188 S-Bit-Serial | 16 bit | 5 | .51 $\mu$s | 2.0 MHz |
| Altera 81188 Parallel | 8 bit | 5 | .156 $\mu$s | 6.4 MHz |
| Altera 81188 Parallel | 16 bit | 14 | .304 $\mu$s | 3.3 MHz |
| CLAy31 S-Bit-Serial | 8-bit | 1 | .421 $\mu$s | 2.4 MHz |
| CLAy31 S-Bit-Serial | 16-bit | 1 | .84 $\mu$s | 1.2 MHz |
| CLAy31 Parallel | 8 bit | 3 | .187 $\mu$ | 5.3 MHz |
| LD LMU08 | 8 bit | 2 | .9 $\mu$s | 1.1 MHz |
| LD LMU18 | 16 bit | 2 | .9 $\mu$s | 1.1 MHz |
| Altera 81188 Fast Parallel | 8 bit | 1 | 1.76 $\mu$s | 567 KHz |
| Xilinx 4010 Fast Parallel | 16 bit | 2 | 4.8 $\mu$s | 208 KHz |
| Xilinx 4010 Constant ROM | 8 bit | 2 | .049 $\mu$s | 20.5 MHz |
| Xilinx 4010 Constant ROM | 16 bit | 5 | .1 $\mu$s | 10.0 MHz |
| LD LF43881 | 8 bit | 3 | .033 $\mu$s | 30 MHz |
| PDSP16256/A | 16 bit | 2 | .08 $\mu$s | 12.5 MHz |

### 3.1 20-tap FIR filter

Performance numbers for a 20-tap FIR filter appear in Table 3. The table entry labeled *TMS320C5X* refers to the popular 16-bit fixed point C5X DSP processors manufactured by Texas Instruments. The benchmark listed is for a C5X with a 35 ns cycle time and a 57 MHz external clock rate [4]. The data throughput rate is less than the inverse of the computation time (1.0 $\mu$s) due to the overhead of executing instructions to set up the filter operation and moving data on and off chip.

The entries labeled *Altera U-Bit-Serial* refer to the use of unsigned bit-serial multipliers to build the 20-tap filters while those labeled *Altera S-Bit-Serial* refer to the use of signed bit-serial multipliers. Mapping inefficiencies for signed bit-serial arithmetic resulted in an increase in system chip count for the signed filters by factors of 3 and 2.5 respectively for the 8- and 16-bit 20-tap FIR filters.

The entries labeled *Altera Parallel* refer to the use of signed multipliers synthesized from VHDL, chosen over the fast adder versions (see Table 1) since the fast adder versions create routing difficulties when multiple multipliers are placed on a chip due to their extensive use of the special logic.

The *CLAy31* bit-serial entries refer to results extrapolated from a signed bit-serial FIR filter design on the CLAy architecture proposed by design engineer Raymond Andraka [1]. The *CLAy31* parallel entry is for the estimated performance of an 8-bit signed parallel version of the filter on the CLAy31 FPGA.

The *LD LMU08* and *LD LMU18* entries refer to the use of custom multiplier chips from Logic Devices in conjunction with an FPGA to implement the filter. The FPGA is used to implement the necessary data delays, data path, multiplier chip control, and the product accumulation required for the multiply-accumulate loop of the FIR filter. Again, a 10 ns on-off chip delay time was assumed. For comparison to equivalent implementations using 1-2 FPGAs with one FPGA being possibly dedicated to implementing the multiplier (16-bit version only) the entries labeled *Altera Fast Parallel* and *Xilinx Fast Parallel* were included.

The next entries in the table present the results for the Xilinx constant coefficient distributed arithmetic multipliers discussed previously. The final entries list results for two custom FIR filter ASICs, the Logic Devices *LF43881* 8x8 bit Digital Filter and the Gec Plessey *PDSP16256/A* Programmable FIR filter.

#### 3.1.1 Comparisons and conclusions

Comparing all of the listed filter implementations it can be seen that the ASIC-based implementations obtain the highest performance. Their performance, however, is nearly matched by the Xilinx-based constant multiplier implementations. This clearly indicates the advantage of the use of the distributed arithmetic approach to multiplication. Using this approach the 8-bit and 16-bit versions of the filter obtain speedup factors of 28 and 13 respectively over the DSP processor. The disadvantage of this approach is the need to implement all of the multiplications in parallel since each multiplier is a constant multiplier and is hence dedicated to a particular filter tap. This results in a larger chip count for the 16-bit filter (5 compared to 2 for the ASIC).

The only systems that performed worse than the DSP processor were those using only a single FPGA-based multiplier to perform the entire filter loop (entries labeled *fast parallel*). In these systems a single multiplier is used to compute the entire 20-iteration multiply-accumulate loop of the filter. This method most closely approximates the method used in the DSP processor for performing the filter but results in inferior performance due to the difference in speed of the FPGA-based multipliers and the VLSI-based multiplier on the DSP processor. Hence, when a custom VLSI multiplier chip is used in conjunction with an FPGA (table entries labeled *LD LMU08* and *LD LMU18*), this architecture again surpasses the performance of the DSP processor.

**Table 4.** Complex Radix-4 FFT Performance

| System | Precision | # of Chips | Computation Time | | |
|---|---|---|---|---|---|
| | | | 64pt | 256pt | 1024pt |
| TI TMS320C5x | 16 bit | 1 | 108$\mu$s | 617$\mu$s | 3.84ms |
| PDSP16116/A | 16 bit | 2 | 11.52$\mu$s | 61.4$\mu$s | .307ms |
| PDSP16510A | 16 bit | 1 | - | - | 98$\mu$s |
| Sharp LH9124 | 24 bit | 2 | - | - | 80.7$\mu$s |

## 3.2 Radix-4 FFT

Comparisons for FFTs using a radix-4 FFT algorithm have also been performed and appear in Table 4. The precision listed for the FFTs in Table 4 gives the number of bits each used for the real and imaginary parts of the input data word to the FFT. The FPGA-based implementation made use of one Altera FPGA and the PDSP16116/A 16-bit complex multiplier chip from GEC Plessey Semiconductors. The FPGA is used for the algorithm control and the implementation of the radix-4 butterfly element. The FFT is computed by calculating each column of the radix-4 FFT successively using the same hardware. Faster implementations can be created by using one FPGA and complex multiplier per FFT column.

As can be seen from the table the FPGA implementation achieves speedup factors of 9.4, 10, and 12.5 over the TMS320C5x DSP processor for FFTs of length 64, 256, and 1024 points respectively. The algorithm as implemented is primarily compute-bound and hence further speedups can be achieved through greater parallelism as mentioned above or through the use of a faster complex multiplier.

For comparison with the performance of ASIC-based implementations the table entries labeled *PDSP16510A* and *Sharp LH9124* are included. These are custom FFT processor ASICs with clock rates of 40 MHz each. The PDSP16510A and LH9124 obtain performance speedup factors of 3.1 and 3.8 respectively over the FPGA FFT processor for a 1024-point FFT.

Using the external multiplier chip along with an FPGA provided an order of magnitude increase in performance over the TI DSP chip. Additional performance increases are possible with ASIC-based systems, however, the performance of ASIC-based systems can be approached or surpassed by the FPGA implementation if one chip set is used per FFT column. For example, using one PDSP16116/A chip and FPGA per FFT column a 1024 point radix-4 FFT could be performed in $307/5 = 61.4\mu s$.

# 4 Conclusion

The results of implementing multipliers on FPGAs presented in Table 1 indicate that for most types of multipliers FPGAs are significantly slower than custom chips. Thus, in order for FPGAs to obtain a performance increase over DSP processors and ASICs, extensive specialization and concurrency must be employed. The distributed arithmetic approach to multiplication was shown to be a method of specialization giving large performance increases in applications where constant multiplications can be applied. The results for the 20-tap FIR filter and Radix-4 FFT indicated that an order of magnitude performance increase over that possible with a DSP processor is not unreasonable for a FPGA-based DSP system. This can be considered a sufficiently significant increase to warrant further application of FPGAs to DSP. In addition, FPGAs provide a reconfiguration advantage over ASICs. Limited flexibility is possible with proper ASIC design but FPGAs have the ability to completely change in function and I/O through reconfiguration. This makes it possible to customize a design through specialization and increased concurrency to achieve best case performance and to reduce costs through the amortization of the silicon over many different applications.

# References


1. Raymond J. Andraka. FIR filter fits in an FPGA using a bit-serial approach. In *Third Annual PLD Design Conference and Exhibit*, 1993.
2. N. W. Bergmann and J. C. Mudge. Comparing the performance of FPGA-based custom computers with general-purpose computers for DSP applications. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 164–171, Napa, CA, April 1994.
3. Kenneth David Chapman. Fast integer multipliers fit in FPGA's. *EDN*, page 80, May 12 1994.

4. Semiconductor Group. *Digital Signal Processing Products and Applications Primer*. Texas Instruments, 1991.
5. Kai Hwang. *Computer Arithmetic: Principles, Architecture, and Design*. John Wiley & Sons, 1979.
6. R.F. Lyon. Two's complement pipeline multipliers. *IEEE Transactions On Communications*, pages 418–424, April 1976.