FPGAs:

THE HIGH-END ALTERNATIVE FOR DSP APPLICATIONS

By Dr. Chris Dick

Engineers have been using field programmable gate arrays (FPGAs) to build high performance DSP systems for several years. FPGAs are uniquely suited to repetitive DSP tasks, such as multiply and accumulate (MAC) operations because they can perform these repetitive operations in parallel. As a result FPGAs can vastly outperform DSP chips, which perform operations in an essentially sequential fashion. In this article Chris discusses how FPGA hardware can be used to augment the processing power of traditional instruction-driven DSP chips, while maintaining the flexibility and the upgrading ability of software-implemented DSP algorithms.

FPGA technology is continuing to advance rapidly, in both performance and density. Today's leading-edge FPGA chips (which offer two-million system gates of logic density) can perform 27 billion MACs/sec, compared with 1.6 billion MACs/sec for the fastest DSP chip currently on the market. By the year 2002 engineers will have access to FPGAs with 10 million gates that can process 500 billion MACS/sec.

However, high-performance, FPGA-based designs have come at a cost. DSP programmers (who live in a software world) are being forced to leap into the world of electrical engineering (and learn a new vocabulary of flip-flops, gates and VHDL code) in order to use FPGA technology.

Fortunately, the job has now gotten much easier because of the availability of:

- DSP algorithms, in the form of *intellectual property* (IP) — also known as *cores*
- tools for incorporating this intellectual property into FPGA-based designs

Intellectual property

A growing number of intellectual property modules that perform DSP functions are now being made available for FPGAs. These predefined modules (whose parameters can be tuned as necessary) provide a variety of standard DSP algorithms, such as...

- filters
- correlators
- sine/cosine building blocks
- transforms
- memories
- math functions

...and can be used to support a wide variety of applications, such as:

- communications
- imaging
- multimedia
- video applications

These cores make it much easier for DSP system designers to use FPGA technology, and to reduce their time-to-market.

Tools for incorporating intellectual property

Tools are also making it easier to incorporate this intellectual property into FPGA-based systems. In addition, traditional system-level modeling tools (already familiar to DSP designers) are soon expected to support FPGAs.

FPGAs can be used to improve system performance

FPGAs also allow system designers to greatly improve the performance of DSP-chip-based systems. For example, FPGA finite impulse response (FIR) filter cores can process hundreds of taps at Msample/sec rates. When used in wireless base stations, test equipment, and image processing systems FPGAs can improve performance levels by at least an order of magnitude. That's because FPGAs permit the design of pipelined data flow architectures, where the data flows from one processing unit to the next inside the FPGA, with minimal signal loading and with no overhead for instruction-fetching or data fetching.

FPGAs can be used to lower system power dissipation

The use of FPGAs (instead of clusters of DSP chips) greatly reduces power dissipation, and provides much higher levels of integration — clear advantages for designers of portable equipment.

Much of the power dissipation of any system is expended in driving the signal traces on printed circuit boards that are used to interconnect the components. For example, systems based on DSP chip technology must drive heavily loaded buses that are used to connect the DSP chips to the memory chips. The extra clock cycles needed to fetch the instructions and the operands from off-chip memory (over these buses) further adds to the total power requirements needed to execute an algorithm.

Because the power consumption of a chip is directly proportional to its clock frequency, the ability of an FPGA to split an incoming data stream and process it as several parallel data streams (at correspondingly lower clock rates) also becomes an important part of the total power equation.

FPGAs can be used to design reconfigurable systems

The rapidly evolving telecom industry has been quick to capitalize on the flexibility and the high performance that FPGA technology provides. Nearly 50% of all FPGA production currently finds its way into telecom and networking equipment of one sort or another, including:

- wireless base stations
- routers
- switches
- modems

FPGAs provide the flexibility needed to track evolving standards (such as WCDMA) and to deal with fluid standards (such as ADSL — asynchronous digital subscriber line).

Take a look at the circuit boards within any wireless base station and you will almost certainly find a programmable device. Programmable logic was first used just to replace glue logic, but it has now become the dominant hardware technology in many wireless base stations.

The digital components found in a base station include:

- DSP chips
- microcontrollers
- FPGAs
- RISC microprocessors
- RAMs
- communication chip sets (such PCI and Ethernet interfaces and HDLC controllers)

FPGAs:

All of these components (except for the communications chip sets) can be completely reprogrammed in the field. FPGAs allow the design of base stations in which the vast majority of the system can be reconfigured remotely *after* deployment — not just the software, but the hardware as well!

This allows a system's functionality to be incrementally upgraded throughout its life cycle. Changes might be required for several reasons, including:

- fixing bugs
- adding new functionality to the system
- adapting the system to evolving standards

It is the last point that is key to keeping up with the evolution of wireless systems.

Third-generation digital wireless systems

Second-generation digital wireless systems (such as IS-95, which replaced the old analog networks) were designed primarily for voice traffic. However, the growth in the Internet has now created a demand for wireless networks with enhanced *data* handling capability.

The third generation of digital wireless technology (referred to as G3) will provide 2-Mbit/sec throughput. With the recent agreement between Ericsson and Qualcom, it now appears likely that the ITU will adopt a CDMA-based standard for G3 systems. The adoption of such a CDMA-based standard would allow wireless network operators to provide a wide range of service levels. However, it would complicate the design of base stations and terminals, because they would need to support multiple modes of operation.

The flexibility needed to handle these various modes of operation can obviously be provided by reprogramming the DSP chips and the general-purpose microprocessors in the base stations. However, these components might not be able to provide the required level of *performance.* The only alternative would be to supplement these chips with customized hardware to perform the most compute-intensive signal processing tasks. FPGAs can provide this hardware, while offering the same degree of reconfigurability as software.

As an FPGA supplier, Xilinx is currently working with several base station manufacturers to help develop the third-generation of wireless communications systems. With the Ericsson-Qualcom agreement, it is almost certain that the next generation wireless standard will allow three different modes of operation. If wireless equipment manufacturers wait for all three of these modes to be fully standardized, network operators will find it impossible to deploy new network hardware in time to meet the forecasted demand.

To prepare for this demand, engineers are being forced to design tomorrow's base stations while the specifications are still evolving. FPGAs have become a key technology because they allow engineers to begin board designs early, and then amend the functionality at a later stage of the project — without having to relayout and remanufacture the printed circuit boards.

Normally in a system as complex as a base station, the FPGAs are configured under the supervision of a microprocessor or a DSP chip. The microprocessor (or DSP) receives the FPGA configuration data through:

- a local I/O port
- a control channel within the wireless network

This data is then held in the system RAM, and eventually downloaded:

- to each FPGA in turn, through a serial chain
- to all FPGAs at the same time, in parallel
- to the various FPGAs, using some combination of these two methods

Note: Most FPGAs cannot be partially reprogrammed. Unless the system is implemented with an FPGA that can be partially reprogrammed (such as the Xilinx Virtex FPGA) care must be taken to ensure that all traffic is rerouted (and the FPGA is taken off line) before any reconfiguration of the FPGA begins.

As the capacities of FPGAs increase, more and more of the system functionality will be embodied in each FPGA device. That will eventually make partial reconfiguration mandatory. Reconfiguration of selected functions within each FPGA will also allow the FPGA silicon to be dynamically adapted to the constantly changing demands of message traffic. The resulting efficiencies will allow reductions in:

- physical system size
- system power consumption
- the cost of the FPGAs

FPGA architectures

There is currently a wide range of FPGA products being offered by many semiconductor vendors, including:

- Xilinx
- Altera
- Atmel
- AT&T

The architectural approaches used in these FPGAs are as diverse as their manufacturers. However, some generalizations can be made. Most FPGAs are organized as:

- an array of logic elements
- programmable interconnections between the logic elements, the I/O pins, and other resources such as on-chip memory

The logic elements

Each logic element typically consists of:

- 1 or more *n*-input RAM-based look-up tables, where n is between 3 and 6
- 1 to several flip-flops

Configuring the FPGA

The configuration and interconnection of the logic elements, the I/O pins and the other resources inside the FPGA is accomplished by downloading a bit stream into a static RAM *configuration memory* inside the FPGA. This bit stream defines:

- the functionality of each of the logic elements
- the internal routing between the logic elements

Different applications can be supported with the same FPGA by simply reconfiguring the FPGA with appropriate bit streams.

The Xilinx Virtex series

To take a specific example, consider the Xilinx Virtex series of FPGAs [2]. The logic elements in this FPGA (called *slices)* consist of:

- two 4-input look-up tables (LUTs)
- two flip-flops
- several multiplexers
- some additional silicon support

Reprinted from DSP Engineering / Spring 2000

Note: The additional silicon support allows for the efficient implementation of *carry-chains* which handle the carry bits when building high-speed adders and subtracters. There is also silicon support that permits the construction of compact, high performance shift registers that can be used to implement very impressive filters, and various types of transforms.

Two of these slices form a *configurable logic block*, as shown in Figure 1. The logic block is the basic unit that is used to build the logic matrix.

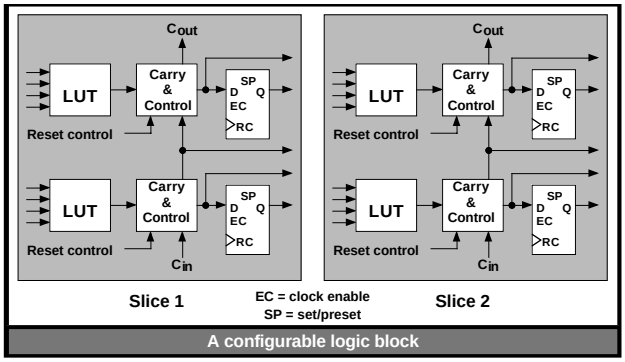

Figure 1

The current generation Xilinx Virtex FPGA family of devices offers 768 to 32,448 logic slices. In addition, they have from 8 to 280 true dual-ported *block memories.* Typical clock frequencies for these devices are in the 100 to 200 MHz range.

In addition to the dual-ported RAM blocks, Xilinx XC4000 and Virtex [2] devices include look-up tables (LUTs) in each logic element, which can be used as memory — either as ROM or as RAM. By distributing the stored data throughout the FPGA (in the LUTs) effective memory bandwidths in the range of many tens of Gbytes/sec can be provided.

Designing a complex multiplier

Complex-valued (real + imaginary) arithmetic is used in a wide range of DSP applications. In fact a widely-used benchmark for a DSP chip is its capacity to perform complex arithmetic — and in particular its capacity for performing complex multiplications. For example, DSP Architectures' DSP-24 [3] can perform a 24x24 complex multiply in 10 nsecs, when operating at 100 MHz.

Note: A 24x24 complex product is generated by multiplying a 48-bit *complex* number (a 24-bit real value, plus a 24-bit imaginary value) by another 48-bit complex number (a 24-bit real value, plus a 24-bit imaginary value).

Implementing a complex multiplier with an FPGA

One method for computing a complex multiplication involves:

- 4 real multiplies
- 1 addition
- 1 subtraction

A 24x24 pipelined real multiplier requires 348 logic slices. A 48-bit adder requires 24 slices, and a 48-bit subtracter requires 24 slices. Thus, a 24x24 complex multiplier can be constructed with: (4 x 348) + 24 + 24 = 1440 slices.

This is only 12% of the logic provided by a Virtex XCV1000 FPGA.

The 4 multipliers, the adder and the subtracter will each operate at a clock frequency in excess of 100 MHz. This allows a complex product to be computed in 10 nsec, which is already competitive with the DSP-24 benchmark. Using a faster grade FPGA permits even higher computational rates.

To achieve even higher performance, several of these complex multipliers could be implemented in a single Virtex chip, and the incoming data stream could be *split* into multiple data streams, with each stream being directed to a different multiplier.

For example, by paralleling 5 complex multipliers, 5 complex products could be computed every 10 nsecs — an effective rate of one product every 2 nsecs. This design would consume about 59% of a Virtex XCV1000 FPGA.

Providing adequate I/O bandwidth

It is all very well to design a parallel data path like the one above. However, in order to be useful, all five of these multipliers must be kept busy 100% of the time. In order to do that, you must supply each multiplier with a new pair of complex input operands on each clock tick.

To provide the I/O bandwidth needed to feed several multipliers simultaneously, the XVC1000 has 512 user-defined I/O pins, which are configurable, and support a wide range of signaling standards.

An alternative complex multiplier design

An alternative method for implementing a complex multiplier would be to use a single time-shared 24x24 real multiplier. This time-shared approach would require about 450 slices (or 3.7 %) of an XCV1000 — or 15% of an XCV300.

Since each complex multiplication requires 4 real multiplication operations, the computation of each complex product would require 4 clock ticks. With a 100-MHz system clock, a new complex product would be computed every 40 nsec. This is still 2.5 times as fast as the DSP-24 benchmark figure.

The ability to architect a custom pipelined data processing path with any degree of parallelism allows FPGA hardware to satisfy the performance requirements of virtually any application.

Using FPGAs to construct digital filters

The Finite Impulse Response (FIR) filter (shown in Figure 2) is one of the basic building blocks common to nearly all DSP systems. The output sample stream is generated by convolving the input sample stream with N filter coefficients. (This operation is referred to as an *inner-product* or *vector dot-product*.) N multiplications and N-1 additions are required to compute each value in the output stream.

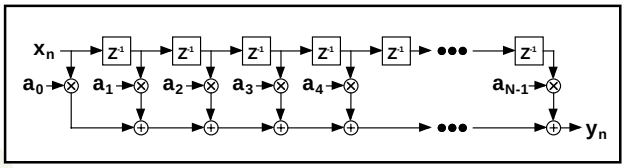

Figure 2

The filter coefficients define the frequency response of the filter. In applications that require a large number of filter coefficients, or high sample rates (or a combination of both) the arithmetic workload can be quite substantial.

DSP chips are often used to implement real-time filters. A higher performance (but less flexible) alternative is an ASIC design. A more recent approach is to exploit the parallelism that FPGAs can provide.

FPGAs:

Implementing FIR filters using FPGAs

There are many ways to implement FIR filters with an FPGA. The most obvious approach (although less than optimal) is to use an architecture similar to the one typically found in an ASIC or a DSP, which employs a single *time-shared* multiply-accumulate (MAC) unit. Since many signal processing engineers are familiar with ASIC and DSP implementations, we will use this as the starting-point for discussing FPGA implementations of FIR filters.

The MAC-based approach

An inner-product computation can be accelerated by using multiple MAC units. In fact, this is a common approach, used by the more recent digital signal processors — both DSP chips and ASICs. This same method can be used in an FPGA implementation. However, an FPGA designer has virtually complete control of the silicon, and can decide how much of that silicon is to be allocated to the inner-product engine.

Implementing an FIR filter with a traditional DSP chip architecture

To provide a reference for evaluating the performance of a MAC-based FPGA-implemented FIR filter, let's first consider the MAC-based DSP-24 digital signal processor. Using...

- a clock frequency of 100 MHz
- 24-bit data
- 24-bit coefficients

...the DSP-24 can compute a real filter tap in 5 nsecs, and a complex tap in 10 nsecs.

Implementing a MAC-based FIR filter with an FPGA

At the heart of most MAC-based engines is a *real multiplier.* A 24x24-bit multiplier is needed to process 24-bit input samples and coefficients. Using Xilinx Virtex FPGA technology this multiplier can be implemented using 348 logic slices. This is 11% of a low-density device, such as the XCV300 [2].

To construct a complete MAC unit, an *accumulator* must be provided on the output of the multiplier. As products from the multiplier are accumulated, the number of bits in the result might exceed 48 bits. In fact, the maximum number of bits required in the accumulator depends upon the number of products that will be accumulated — an application specific number.

When implementing a MAC with an FPGA, the architect is free to choose the precision of the accumulator. For this example we will specify a 56-bit-wide accumulator. The accumulator requires 28 logic slices.

Excluding the small amount of control logic needed for address generation, and for scheduling of the arithmetic unit, this MAC engine can be implemented with about 348+28 = 376 logic slices.

In addition to this MAC engine, a complete FIR filter requires storage for:

- the filter coefficients (N values)
- the input sample history buffer (N values)

There are several memory resources that can be used to provide this storage in a Virtex FPGA, including block RAM or distributed RAM (LUTs).

For large filters (with many taps) the block RAM is needed to provide enough storage for both the input samples and for the filter coefficients. For smaller filters (with fewer taps) the input samples (and/or the coefficients) can be stored in distributed RAM, which is provided by the 16x1 LUTs that are part of each logic element in the fabric.

For example, a 16-tap filter using 24-bit input samples and 24-bit coefficients, requires a total of 24 slices to provide both the filter memory and the coefficient storage. A clock frequency of 100 MHz provides a multiply-accumulate rate of 10 nsecs/tap, or 100 MMACs of performance.

The configurable nature of FPGAs would permit a designer to exploit the high degree of potential parallelism in many DSP algorithms, to achieve even higher levels of performance. In this case, two MACs could be paralleled in a single filter to provide a 200-MMAC throughput.

This dual-MAC design requires about 3% of an XCV1000 FPGA, and computes the equivalent of a MAC every 5 nsecs. Additional MAC units could be employed to further reduce the effective MAC cycle time.

Parallel execution in DSP chips

Many current DSP chips have multiple execution units for performing arithmetic operations. These execution units often include several MAC units, for accelerating filter computations. Programmers can use parallelism (up to a point) to increase the performance of MAC operations. However, the degree of concurrent execution is limited by the number of MACs in the DSP chip, often preventing the programmer from fully exploiting all of the potential concurrency in the algorithm.

In many DSP chips the effective number of MAC units can be increased, at the cost of lower *precision* in the MAC calculations. For example, one current DSP chip can compute two 80-bit real MACs (32-bit data and coefficients) or eight 40-bit MACs (16-bit data and coefficients) per clock cycle. The latter figure corresponds to a performance of 2 GMACs (GMACs) at a clock frequency of 250 MHz.

Parallel execution in FPGAs

In contrast, FPGA hardware allows the designer to allocate silicon resources to fully exploit the concurrency in an algorithm. For example, if an algorithm performs 14 multiplies, and then sums all of the products, an FPGA could be configured to perform all of the 14 multiplications in parallel, using 14 traditional MAC-style inner product engines.

Suppose each of these engines supports 16-bit data, 16-bit coefficients, and a 40-bit accumulator:

- Each of the 16x16 multipliers would require 168 logic slices.
- Each of the 40-bit accumulators would require about 20 slices.

The simultaneous outputs of these 14 MAC units could then be combined, using an *adder-tree.*

The 14-MAC units and the adder tree could be implemented with about 2,892 slices. This represents 61% of an XCV400 FPGA [2]. When driving these units with a 150 MHz clock, the 14 MAC units would provide 2 GMACs of performance.

FIR filters: An alternative design approach, using distributed arithmetic

The performance of the highly-parallel MAC architecture described above is indeed impressive. However, we should ask

Reprinted from DSP Engineering / Spring 2000

FPGAs:

ourselves if this DSP-chip-inspired architecture is the optimal architecture for solving this problem with an FPGA. In many cases the answer to this question is "no".

A wide variety of very creative optimizations of DSP computational hardware have been reported in the open literature over the last few decades. However, these approaches can be very application specific, and require customized hardware.

Unfortunately, general-purpose DSP chips must be designed with architectures that perform reasonably well over a wide range of applications. The resulting architecture is a compromise. Novel signal processing algorithms (that might be useful for specialized DSP applications) cannot be optimally supported with these general-purpose DSP chips.

However, because FPGA hardware is configured with internal SRAM memory, FPGA chips can be configured to support a wide range of applications, simply by downloading the appropriate configuration bit-stream. FPGAs are like miniature silicon foundries, with extremely short turn-around times. This allows a system architect to design a custom architecture for any particular application.

Let's take the case of computing an inner-product. One optimized approach (which was first published in the open literature by Peled and Liu [1]) is called distributed arithmetic. For an excellent tutorial on the use of distributed arithmetic for FIR filtering, IIR filtering, and FFTs, the reader is referred to the article by White [4].

A generic model of a distributed arithmetic filter is shown in Figure 3. Distributed arithmetic calculations are accomplished with:

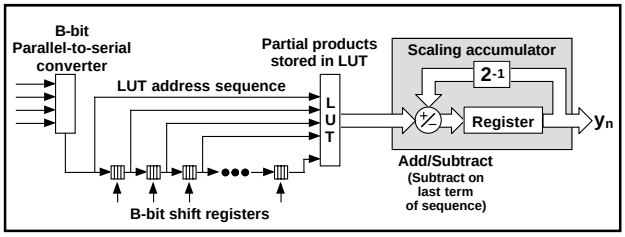

Figure 3

- table look-ups
- additions
- subtractions

All three of these operations are well suited to FPGA implementation.

One very interesting property of distributed arithmetic filters is that the filter throughput is not dependent upon the filter length. It depends on the input sample precision, instead. This is shown by the linear plots in Figure 4. For a given input sample precision (B) the processing rate remains constant, independent of the number of filter taps.

For example, with a 100 MHz clock and an input sample precision B = 12, the filter processing rate is 8.333 MHz, regardless of whether the filter length (N) is 10, 20 100, or 200. For N = 200, this is an effective computation rate of one MAC every 0.6 µsecs — or 1.7 GMACs/sec.

Taking advantage of symmetrical filter coefficients

For many applications the set of filter coefficients is symmetrical. This symmetry can exploited to reduce the logic required to implement the filter.

The computation rate is reduced slightly. For a 100 MHz clock, and 24-bit input samples, the computation rate will be 4 MHz. Again, the computation rate is constant, regardless of the number of taps. So, for a 200-tap filter this still produces an impressive 800 MMAC/sec computation rate.

Increasing the speed of multiplication using parallel distributed arithmetic

Distributed arithmetic involves the computation of many partial products (in parallel) and then the summation of all these partial products, to compute the overall product. However, storage must be provided (in the distributed LUTs) to hold all of the multiplication tables used to accomplish the partial product multiplications.

The number of required LUTs can be minimized by using a shift register to shift each multiple-bit input value into the FPGA, one bit at time. The 1x1 partial products are then computed one bit at a time, and accumulated inside the FPGA until the final output is generated.

However, processing the data in this serial fashion (one-bit-at-a-time) produces rather modest computational rates. When each input variable is B bits in length, B clock cycles are needed to complete a single inner-product calculation.

Speed can be improved in several ways. One approach is to split each input word into L subwords, and then shift each of these subwords into the FPGA in parallel. (The *function generators* in FPGAs can be used to provide the required shift registers.) The tap points of these shift registers are then used as *address inputs* to the LUTs, which compute the partial products. This method requires L-times as many LUTs, and so the speed-up comes at a cost of a linear increase in the multiplication table storage requirements. For example, an 80-tap filter using 12-bit input samples and coefficients can be constructed with 2800 logic slices, and will support a sample rate of 150 MHz. This is equivalent to 12 billion MACs/sec.

An alternative to using the distributed LUTs to is to employ the block memory that is available in the more-recently-introduced FPGAs. For example, consider building a symmetrical 70-tap FIR

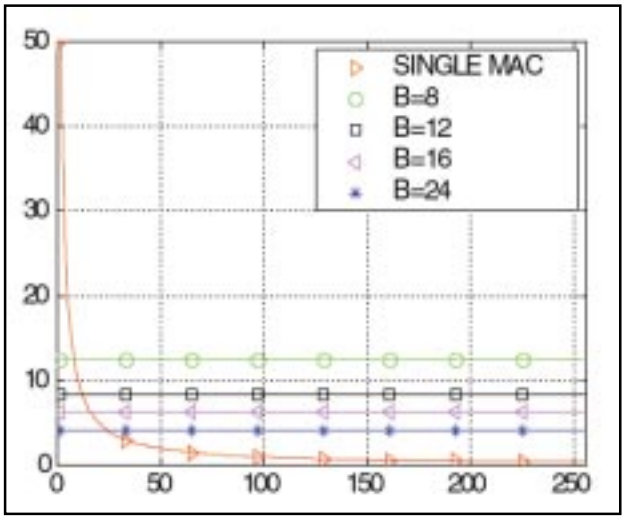

Figure 4

filter, in which there are only 35 unique filter coefficients. With 70 taps, a serial one-bit-at-a-time LUT approach would require the storage of a huge number of partial product terms. The number of partial products becomes much smaller if the 35 tap points on the shift register are partitioned into 5 terms:

- 4 groups of 8 bits
- 1 group of 3 bits

Now only four 256-entry multiplication tables, and one 8-entry multiplication table is needed to compute the partial products. These 5 multiplication tables can be stored in on-chip block memory, and a simple adder tree can be employed to combine their outputs.

Fourier transforms

The discrete Fourier transform (DFT) is one of the most common operations performed in signal processing systems, and it is typically implemented using the fast Fourier transform (FFT) algorithm.

A state-of-the-art DSP chip can perform a 1024-point complex FFT on 16-bit complex samples in about 66 µsecs. Additional time (~9 µsecs) is typically needed to perform a bit-reversal permutation, if required.

Using a standard radix-4 Cooley-Tukey algorithm, and exploiting only a modest amount of parallelism, the same FFT can be performed in a Virtex FPGA in 41 µsecs — and this includes the bit-reversal operation.

While these comparisons of arithmetic performance are useful, there is still one extremely important consideration that is often ignored when comparing FPGA performance to DSP chip or ASIC performance. This is the *I/O bandwidth*.

With the exception of a few dedicated pins, nearly all of the pads on an FPGA are available for I/O. This provides extremely high I/O bandwidth in FPGA-based signal processing designs.

The large number of I/O pins (combined with the multiple banks of internal memory) allow an engineer to employ a high degree of parallelism in an FPGA design. In the FPGA-based FFT design shown in Figure 5, separate memory banks are used for the input/working memory buffer, and for the output memory buffer.

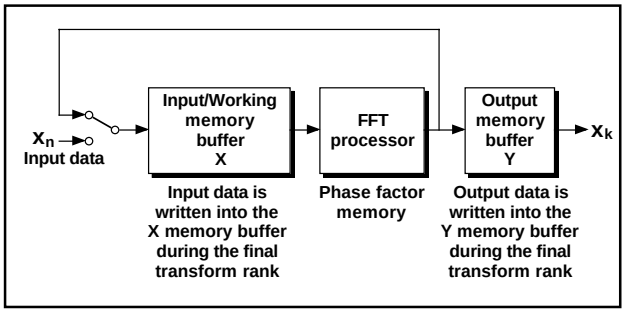

Figure 5

This permits I/O to be performed concurrently with the FFT computation. So, in the case of the FPGA design, the 41-µsec execution time includes all input, output and computational operations.

Designing higher performance FFTs through parallelism

The FFT algorithm provides many opportunities for exploitation with parallelism. Current FPGAs provide the I/O bandwidth, the on-chip memory and the logic resources to build highly concurrent single-chip FFT engines.

For example, using a direct partitioning of the standard radix-4 Cooley-Tukey FFT algorithm, one computation unit could be assigned to each column of butterflies. This approach would require a total of 5 butterfly engines, plus the memory buffers between successive butterfly columns.

A 1024-point FFT could be computed in 1024 clock cycles. Using a 100 MHz clock, the transform time is 10 µsecs. If required, more parallelism could be used to produce a single-chip FFT solution with sub 10 µsec computation time.

Conclusion

FPGAs are being employed in a wide variety of signal processing applications because of their:

- superior performance
- low cost
- flexibility
- low power consumption

The telecom industry has been particularly quick to embrace FPGA technology. Nearly 50% of all FPGA chips currently being manufactured are being used in telecom and networking equipment of one sort or another — wireless base stations, switches, routers and modems, to name a few.

The versatility of FPGA technology allows the support of multiple protocol standards. One application might be a universal cellular handset that automatically recognizes different signaling standards (such as GSM, CDMA, TDMA, or AMPs) and reconfigures itself to accommodate the recognized protocol.

The flexibility and the high performance provided by FPGAs also allows engineers to easily track evolving standards (such as MPEG) and to work successfully with fluid standards (such as ADSL).

FPGA technology already represents a significant fraction of currently-deployed signal processing hardware, and we are witnessing an exponential growth in the insertion of FPGAs into digital signal processing systems. This explosive growth is being enhanced by access to FPGA intellectual property (IP) cores from all of the major FPGA suppliers, as well as 3rd-party IP designers. With these resources, system engineers are able to focus on their system architecture, instead of getting bogged down in the details of lower-level modules, such as filters and transforms.

The continuing evolution of communication standards (and the highly competitive pressures in the marketplace) dictate that engineers must begin their design while standards are still evolving. In addition, third-generation wireless standards (and future-generation wireless standards) will need to support multiple modulation formats and air interface standards. FPGAs provide the flexibility to achieve this, while simultaneously providing high levels of performance.

References

[1] Peled and B. Liu, "A New Hardware Realization of Digital Filters", *IEEE Trans. on Acoust., Speech, Signal Processing*, Vol. 22, pp. 456-462, Dec. 1974.

[2] Xilinx Inc., *The Programmable Logic Data Book*, 1999.

[3] DSP Architectures, "DSP-24 Preliminary Data Sheet", Vancouver, WA 98665. (http://www.dsparchitectures.com/dsp-24/dsp-24.htm)

[4] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing", *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.

[5] E. B. Hogenauer, "An Economical Class of Digital Filters for Decimation and Interpolation'', *IEEE. Trans. Acoust., Speech Signal Processing*, Vol. 29, No. 2, pp. 155-162, April 1981.

[6] B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[7] The Mathworks Inc, *Matlab, Getting Started with Matlab*, Natick, Massachusetts, U.S.A, 1999.

[8] The Mathworks Inc, *Simulink, Dynamic System Simulation for Matlab, Using Simulink*, Natick, Massachusetts, U.S.A, 1999.

[9] Xilinx Core Generator System, http://www.xilinx.com/products/logicore/coregen/index.htm

[10] C. H. Dick and F. J. Harris, "Direct Digital Synthesis — Some Options for FPGA Implementation", *SPIE International Symposium On Voice Video and Data Communication: Reconfigurable Technology: FPGAs for Computing and Applications}* Stream, Boston, MA, USA, pp. 2-10, September 20-21 1999.

Dr. Chris Dick *is the manager of the Signal Processing Group at Xilinx Inc. He is responsible for coordinating the DSP IP engineering activities, and is the DSP technical authority at Xilinx. Chris joined Xilinx in 1997 from La Trobe University in Melbourne Australia, where he was a professor for 13 years. He has published more than 60 journal and conference papers, and has been a speaker at numerous international DSP and communications symposiums. Chris holds PhD and bachelor's degrees from La Trobe University, both in electronic engineering.*

If you have questions about this article, or if you would like more information about Xilinx products, you can contact Chris at:

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
Tel: 408-879-5377
Fax: 408-626-6440
Email: chris.dick@xilinx.com