

# FPGAs and DSP

## Design alternatives for DSP solutions

As a designer of Digital Signal Processing systems, you have a large number of choices to implement your solution. Each solution has its strengths and weaknesses. The purpose of this chapter is to introduce Xilinx Field Programmable Gate Array (FPGA) technology and help you understand how FPGAs can be used for DSP system implementation.

In the following sections, you will find a comparison of the implementation of a simple DSP function in both Programmable DSP (pDSP) and Gate Array technology. A brief explanation of Gate array technology is followed by a description of Xilinx FPGA technology. If you are already familiar with Gate array technology, you might want to skip to the last section which highlights Xilinx FPGA features.

### Programmable DSPs

The most common vehicle for implementation of a DSP design is *the programmable DSP* or pDSP. The pDSP is an off-the-shelf part that is essentially a microprocessor tuned to DSP applications. pDSPs are highly flexible because you can program them again and again using a familiar high level language like C. They allow fast design iterations and reduce time to market.

Typically a pDSP contains several functional units to process the signal stream. The designer encodes the algorithm into a program which is executed by the pDSP and is limited to a theoretical maximum data rate based on the speed and the number of multiplier/accumulators in the device. Applications which require several computations must be broken up into a sequential stream of computations. For example, an 8-tap FIR filter requires 8

Figure 1. 8 tap FIR Filter

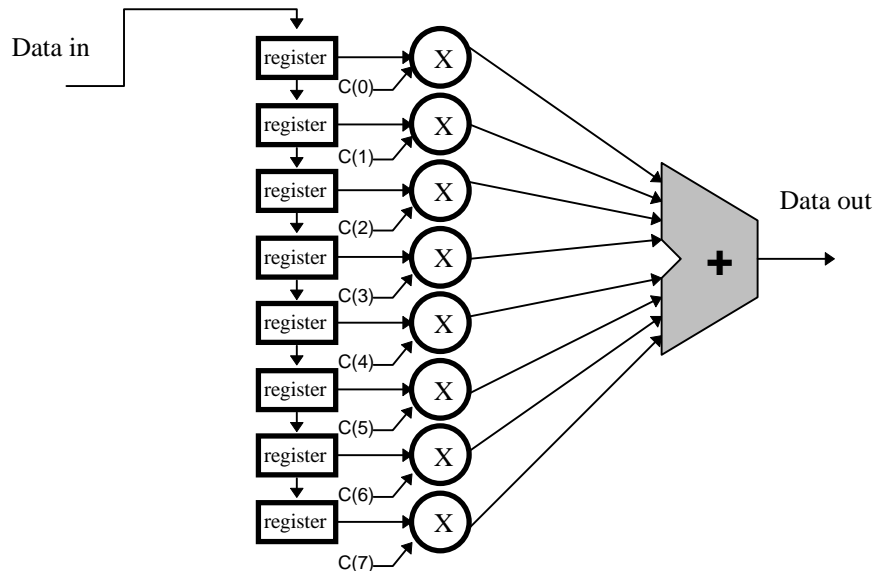


Figure 2- Sample pDSP instruction stream for 8 tap FIR Filter

```

MultAcc Reg(0), C(0)
MultAcc Reg(1), C(1)
      :      :
MultAcc Reg(7), C(7)

```

multiplications and one 8 way addition per data sample. The implementation of this FIR filter might require 8 or more cycles on a pDSP. At each data sample, all eight taps

require multiplication by their coefficients. If the number of taps for this filter were increased, then the number of cycles would also be increased, thereby reducing the data rate.

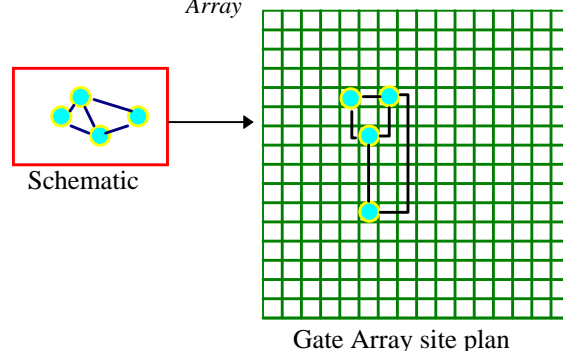
Programmable DSP chips are intrinsically limited in performance. The more you want to do to a data sample, the more cycles you need and the slower your data is processed. One way to overcome this limitation is to employ more pDSP parts to implement the algorithm. Another method is to use Gate Array technology to implement your algorithm in hardware.

### Gate Array solutions for DSP

In contrast to pDSPs, Gate Array technology offers the ability to do many things in parallel. A Gate Array is a custom chip that allows very specific implementations of digital circuits to be constructed. Gate Arrays can be thought of as tracts of open land. You, the designer decide what factories or processing plants will be built on this land and how the materials flow between them. For DSP applications, these factories are arithmetic operators, storage elements and so forth.

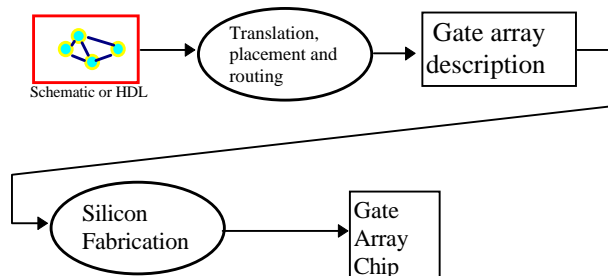
A Gate Array is an open grid of sites which can be occupied by logic cells selected from a library. A Gate Array designer may draw a schematic diagram for the circuit and then rely on automatic tools to place components of his circuit into Gate Array sites. The automatic tools also handle the low level details of connecting the components according to the schematic. Figure 3 depicts a very simplified example of a design circuit being converted to the orderly layout of a Gate Array.

Figure 3- A schematic implemented on a Gate Array



An alternative to using a schematic for the design description is to use a Hardware Description Language (HDL). HDLs are high level descriptions of circuit behavior. These higher level descriptions are often easier to create and understand than schematics and typically look a lot like programming languages. *Synthesis* tools are used to translate the HDL text into a lower level circuit built from the Gate Array's cell library. Figure 4 shows a typical Gate Array design flow where the designer has specified the design with a schematic or an HDL definition.

Figure 4- Gate Array design flow



Gate Arrays offer to the DSP designer a higher degree of parallelism. Multiple execution units can be built that can operate simultaneously. DSP applications are very parallel in

the sense that the data always flows forward and the relationships between intermediate values is well understood at design time. Furthermore, since Gate Array technology allows custom design, you are not constrained to predetermined standards such as operand precision. If your application calls for 11-bit data elements, you build your execution units to be 11 bits wide. By using only the data width you need, you save silicon area and reduce the cost of the device.

Gate Array implementations of many DSP algorithms are very straightforward. For example, the 8 tap FIR filter presented above can be directly translated into a gate array design. Each of the multiply and add operators could be implemented by a region of the gate array. At any time all 8 multiply operations, as well as the multiple operand addition, can be performed simultaneously. If more taps are required, more Gate Array area can be allocated to implement them. Performance improves further when *Distributed Arithmetic* methods are employed.

The primary reason to choose a Gate Array solution is increased performance/cost ratio. A fringe benefit of Gate Array technology is that it can be used to implement more than just DSP circuitry. Other pieces of a multi-chip system can be “swept” into unused portions of Gate Array area (or a larger device could be used) to further reduce chip count and system cost.

The downside of a Gate Array is that it is a custom part. Once designed, it must be custom or semi-custom fabricated at a silicon foundry. Due to the unique nature of the device a custom Gate Array typically requires several weeks to be fabricated from the prototype plans, and due to the expense of the overall process the design must be carefully verified prior to the manufacture of even small quantities. Design flaws found after fabrication require costly and time consuming “spins” of the design.

## **FPGAs - The best of both worlds**

Field Programmable Gate Arrays are a technology which gives the designer a combination of the benefits of a gate array solution and the ease of pDSP design. An FPGA design starts with the same input as a Gate Array design - i.e. a circuit schematic or high level design description. Automatic synthesis, place, and route tools are used to translate the designer's original circuit into an FPGA specific *configuration*.

The big difference between the Gate Array and FPGA design process is that the user specific custom manufacturing process is eliminated.

FPGAs are generic commodity parts and are customized by downloading a user defined configuration in the form of a binary *bitstream*, much the same as you would load a pDSP with its program with a process that typically takes only a few milliseconds

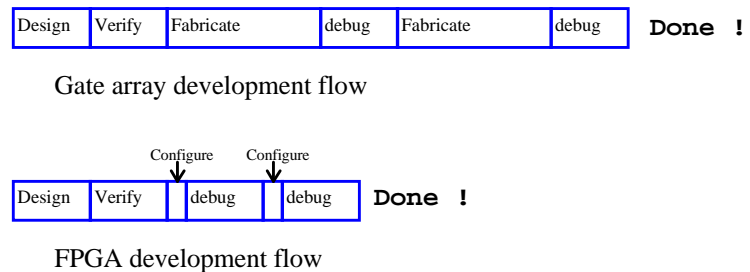
How much of an advantage does the FPGA's ability for direct implementation buy you? The answer really depends on how many tries you think you will need to converge on a correct implementation. Most

DSP designs are part of a complex system.

Experience shows that it is very common for complex systems to go through several design iterations before product completion. Figure 5 compares the development cycle of a new design as implemented on Gate

Array versus FPGA technology. This development cycle includes extra iterations to fix bugs. By virtue of immediate implementation, an FPGA based design solution is ready for delivery much earlier than a Gate Array design.

Figure 5 - comparison of Gate Array and FPGA development cycles



Why, if FPGAs provide such benefits, would anyone use Gate Arrays? The answer is that for very high production volumes, FPGAs do not exhibit as high of a performance/cost ratio as Gate Array or full custom Application Specific Integrated Circuit (ASIC) designs. The lower performance/cost of FPGAs comes in part because FPGAs must sacrifice some silicon area in order to be highly flexible.

However, this should not be a concern to the designer expecting to ramp up to high volume. Xilinx offers a design migration product called *Hardwire* which retains much of the performance/cost advantage of Gate Arrays. Once the designer has converged on a working FPGA design, she can then translate the design to an equivalent Hardwire device without the need to redesign or debug the system.

In summary, Xilinx FPGAs offer the rapid design cycle of programmable DSPs with the flexibility and raw performance of Gate Array products.

Other advantages of FPGAs include:

1. Parts may be reprogrammed over and over. If you want to upgrade your design, you do not need to replace FPGAs, just reprogram them.
2. FPGAs are pre-tested. Traditional Gate Array design methodology requires that you also develop costly manufacturing test suites. This task is not required with FPGAs.
3. FPGAs are a commodity part. Xilinx sells millions of FPGAs annually. This high production volume results in a lower per part cost and those savings are passed on to the customer.
4. FPGAs can be dynamically reconfigured within the system. Sophisticated designers can build systems which adapt to changing conditions by altering the circuit configured within the FPGA. This re-configurable design approach is becoming more and more popular since many systems need to perform several different functions, but never all of them at the same time.

## **Xilinx FPGAs - a closer look**

The Xilinx product line includes a wide variety of FPGA and Complex Programmable Logic Device (CPLD) chips. This DSP Toolbox is geared towards the design of three closely related members of the Xilinx family - the XC4000E, XC4000EX and XC4000LX.

This section focuses specifically on these families supported, but many other Xilinx devices are also excellent platforms for DSP systems.

In the discussion that follows, you will see the features of the Xilinx XC4000 devices that are crucial for the design of high performance DSP systems. Detail is kept to a minimum. For more information you should refer to the application notes.

The XC4000 series of devices possess the following features which enable high performance DSP design:

1. Flexible logic blocks with bit level arithmetic features - allows Distributed Arithmetic implementations of DSP algorithms.
2. Fine grained distributed RAM and ROM - Increases operand bandwidth.
3. A register rich architecture - enables a high degree of pipelining leading to increased performance.

### **The XC4000 CLB - flexibility to support Distributed Arithmetic**

Let's start by looking at the XC4000 series Configurable Logic Block (CLB). These are the basic building blocks of the XC4000 FPGA. The figure below is a simplified diagram of the resources within a CLB. You don't need to understand this diagram in detail, just a few important features.

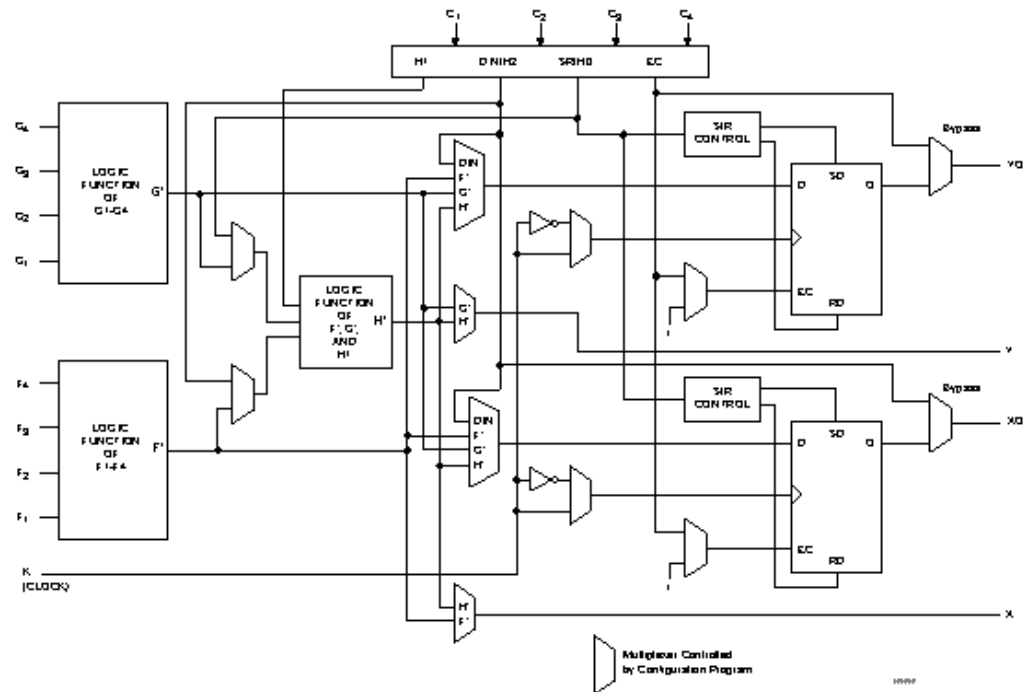


Figure 6 Xilinx XC4000 CLB

One way to look at the CLB is as a two bit functional block. The two Logical Function blocks of the CLB (on the left) can implement any function of its input variables. This includes all of the basic building blocks required to construct any arithmetic function.

Because CLBs can be interconnected in any way, they can be used to construct Distributed Arithmetic implementations of DSP functions. Distributed Arithmetic will not be discussed in detail here, but one important characteristic is that:

***It delivers the performance of a fully parallel circuit within the space of a serial circuit.***

Serial arithmetic circuits are great because they remain small with respect to the size of the operands they process. At most, they grow in linear proportion to the operand size. However, serial arithmetic requires that at least one of the operands be processed one bit at a time, requiring multiple machine cycles per sample.

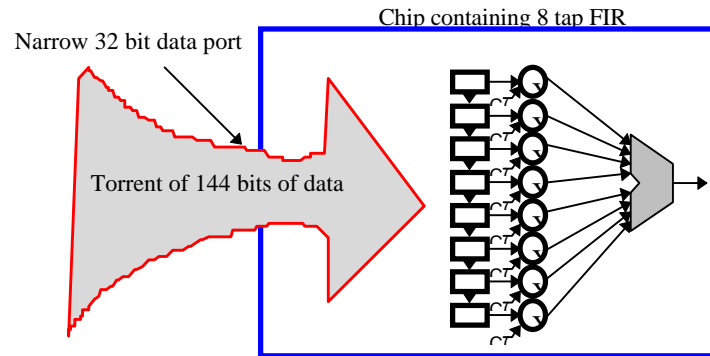
Alternatively, parallel implementations only require a single cycle per sample, but the speed comes at a cost, since parallel multipliers grow in proportion to the square of the operand size.

Distributed arithmetic gives you the benefit of both serial and parallel implementations, resulting in small, fast, efficient circuits. For more information, see the documentation on Distributed Arithmetic.

### Distributed memory increases bandwidth

If you recall from the 8 tap FIR filter example presented at the beginning of this chapter, there are several sources of operand data. Most obvious is the data sample stream being

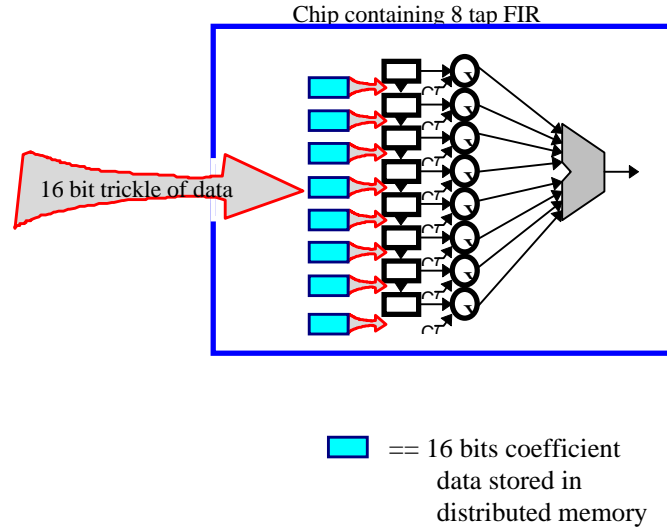
*Figure 7 - IO Bandwidth limitations*



processed by the filter. In addition, there are 8 coefficients entering the block. If all of this data had to come from outside of the chip implementing there could be a significant bottleneck. For example, let's assume that the FIR filter processes 16 bit data and the DSP chip has a 32 bit data interface to the rest of the system. Only two operands could be transferred per cycle. Our FIR filter could only run at 1/5 of its top speed because it would have to wait for its data to squeeze through the narrow input port.

Typically, coefficient data is confined to a small set of values, and it is possible to store the coefficients locally on the DSP chip. The XC4000 FPGA contains a distributed Random Access Memory feature. This feature can be used to overcome the bandwidth bottleneck.

Figure 8 - Distributed memory eliminates the data bottleneck



The use of internal FPGA memory to store raw coefficients is a powerful use of the XC4000 distributed memory feature. As you will discover later in this documentation, distributed memory and distributed arithmetic form a powerful combination. Extremely high performance DSP systems can be implemented using these techniques on Xilinx FPGAs.

### Pipelining increases throughput

Recall from Figure 6 that the XC4000 CLB contains a pair of registers. These registers can be used to *pipeline* a function. Pipelining means that a function such as multiplication is divided into smaller steps. In FPGA technology, smaller steps mean less circuitry between clock steps, which in turn means that the clock can run faster and overall performance increases.

Pipelining comes at an expense - results are delayed some small finite number of clock cycles before they are complete, but this latency is of negligible effect on DSP designs. For most DSP applications the data rate is much more important than any latency.

Pipelining is a natural mode of system optimization on XC4000 designs. Registers are everywhere and are essentially “free”.

## Summary

This section presented a brief overview of Xilinx technology applied to DSP applications. Some of the basic differences between traditional programmable DSP chips and FPGAs were highlighted. A few of the Xilinx FPGA features which are useful in DSP design have been presented. You can learn more about these in the following chapters.

Although there are a lot of ideas presented here, they are a fraction of what that can be done with Xilinx FPGAs. FPGAs are extremely flexible devices and engineers are inventing novel applications every day. If you take the time to review the features of Xilinx devices, you may discover a new approach to solving your design problem.