FPGA TECHNOLOGY IN INSTRUMENTATION AND RELATED TOOLS

J. Serrano, CERN, Geneva, Switzerland

Abstract

Field Programmable Gate Arrays (FPGA) have become an alternative to traditional Digital Signal Processors (DSP) in many applications. In some cases, where high throughput is the main concern, an FPGA-based system may in fact be the only solution to fulfil the requirements. In the area of particle accelerators, FPGAs are used in many contexts, ranging from digital feedback loops for power converters and RF cavities to Digital Signal Processing for beam instrumentation. These designs harness the vast amount of logic resources inside FPGA chips to deliver unprecedented performance through parallelism and pipelining. After an introduction to the internal architecture of FPGAs and the design process, including advanced issues such as floorplanning, we look at two important techniques to implement arithmetic in FPGAs: Distributed Arithmetic (DA) and the COordinate Rotation DIgital Computer (CORDIC) algorithm. The goal is not to exhaust the list of Digital Signal Processing techniques for FPGAs, but rather to illustrate ways in which FPGAs are used to maximize performance.

INTRODUCTION

Programmable logic technology has been a central player in the glue logic and bus interface arena since the 1980's. In the last decade, due to the exponential growth in silicon densities and the maturity of the associated software tools, new segments of the digital design market have found solutions in the FPGA realm. One of the most impressive examples is the growing number of Digital Signal Processing (DSP) systems implemented in FPGAs. Accelerator subsystems, where purely digital applications such as those based on counters and state machines have existed for years, are now benefiting as well from the determinism and the speed of FPGA-based DSP solutions.

While this article presents general ideas applicable to any FPGA manufacturer, the examples and specific terminology, when needed, are those of Xilinx [1] products, with which the author is most familiar.

FPGA INTERNAL STRUCTURE

The architecture of an FPGA chip is essentially a rectangular array of Configurable Logic Blocks (CLB) interconnected by a programmable routing matrix. Figure 1 shows the internals of a generic chip from the Spartan IIE family, chosen for purposes of illustration due to its simplicity. CLBs can implement basic combinatorial functions and the result of these operations can either be routed directly out of the CLB or be clocked in internal CLB flip-flops before going out to the routing matrix.

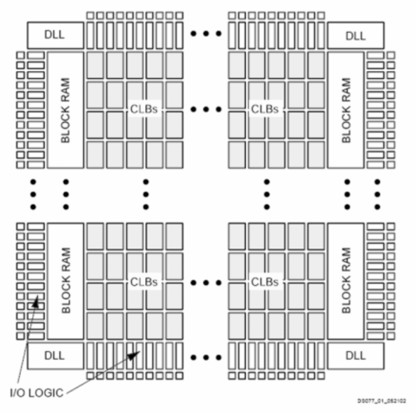

Figure 1. Internal structure of the Spartan IIE family. (Xilinx corp.)

Even in this low-range family, the designer has some extra resources available, namely internal RAM blocks and Delay Locked Loops (DLL) used to manage the phases and frequencies of inter-related internal clocks. Other blocks found in modern FPGA families include, among others, hard-wired Multiply and Accumulate (MAC) units, fast dedicated serial transceivers (useful to cluster several distant FPGA systems in global orbit correction applications) and Digitally Controlled Impedance (DCI) for on-chip high speed signal termination. Figure 2 gives a simplified view of the internals of a CLB.

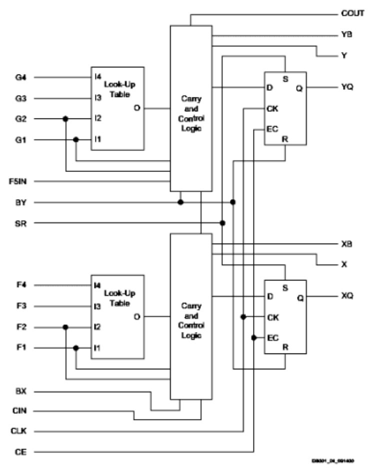

Figure 2. Spartan IIE CLB internal structure. (Xilinx corp.)

The Look Up Tables (LUT) can implement any combinatorial function of four inputs and one output (of the type you would write down using AND and OR operators). An interesting feature is the Carry In (CIN) – Carry Out (COUT) daisy chain, which links each CLB with its two vertical neighbours through dedicated routing resources, independently of the programmable routing matrix. As its name suggests, this daisy chain is used to implement fast carry logic from one bit to the next in big adders.

For high speed DSP applications, the advantages of FPGA solutions with respect to DSP chips should be apparent from Figure 3. In a DSP with a single MAC unit, a 256-tap Finite Impulse Response (FIR) filter can only accept a new data sample every 256 clock cycles, since this is the time needed to calculate an output. With the pipelined design in an FPGA, input samples propagate through one stage every clock cycle generating a new output value, so the filter can accept one input sample per clock cycle.

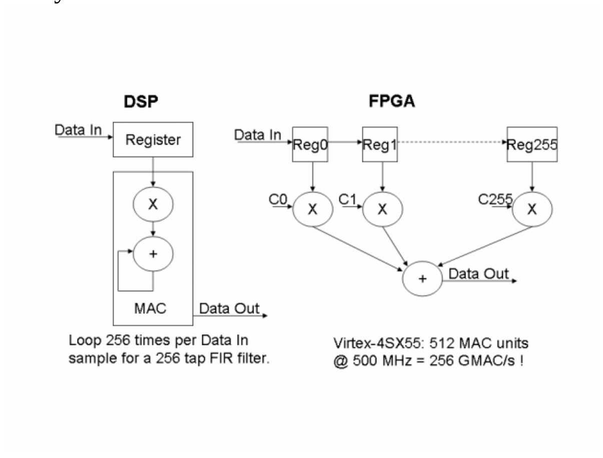
DSP
FPGA
Data In
Register
X
+
MAC
Data Out
Loop 256 times per Data In sample for a 256 tap FIR filter.
Data In
Reg0
Reg1
Reg255
C0
C1
C255
X
+
Data Out
Virtex-4SX55: 512 MAC units @ 500 MHz = 256 GMAC/s !

Figure 3. Architectural difference between DSP chips and FPGAs.

If we compare peak performances of top-of-the-line DSP chips against FPGAs (June 2005), in terms of raw GMAC/s without any architectural consideration, we find that the best FPGA can currently deliver 256 GMAC/s (see Figure 3) while the fastest DSP chip can perform 4 MAC operations per clock cycle in each one of its two multipliers, at a clock speed of 1 GHz, i.e. a total of 8 GMAC/s.

FPGA DESIGN FLOW

We now look at how the use of modern software tools helps the designer turn an idea into working hardware. The design flow can be split into three separate phases.

Design Entry

In this phase, the designer uses a software tool to describe the digital system. As circuit complexity increases, traditional schematic drawing methods are replaced by text-based input using a Hardware Description Language (HDL) such as VHDL or Verilog. These text-based methods harness the good design practices coming from the software world, such as code reuse, modular well partitioned designs, comments in the code, etc. For even higher abstraction, some tools can automatically generate HDL from block diagrams where each block represents a complex functional unit such as FFT, PID Controller, FIR filter, etc.

Synthesis

Synthesis tools typically get HDL as input and find out, in a first phase, what the designer is trying to implement. For example, the VHDL statement *DummyOut <= DummyInA when Selector='1' else DummyInB;* would be interpreted by a synthesis tool as a 2-to-1 multiplexer. The whole design is thus translated from text into so-called Register Transfer Level (RTL), a level of abstraction where a circuit is composed of standard combinatorial blocks (multiplexers, decoders, gates, etc.) and registers made of flip-flops. In a second phase, the tool maps the RTL design into the chosen chip. In our example, the multiplexer will end up mapped into a LUT where only three of the four inputs are used.

Place and route (P&R)

The place and route tool takes a netlist output by the synthesis tool, and maps it geographically onto the chip in two steps. In the placement phase, proper CLBs will be chosen to implement the functions described in the netlist. The subsequent choice of interconnection paths between these CLBs constitutes the routing phase.

As we shall see, designs consist very often of groups of combinatorial logic sandwiched between register banks. Every time a clock tick happens, signals propagate from the inputs of the registers to their outputs and then through the combinatorial path generating a result fed to the inputs of the next register bank. This result should be ready by the time the next clock tick comes if we want the circuit to function correctly. Maximum propagation delay through combinatorial logic therefore determines the maximum clock rate at which a design can work.

In this context, the choice of CLBs in the placement stage is critical for timing performance. A clear example is the case of a 16 bit counter, which involves addition and can therefore take advantage of the vertical carry daisy chain between CLBs. A design where only one such counter is instantiated will be an easy task for the placement tool, which will very likely use vertically adjacent CLBs to map the structure. In a more dense design, however, the different bits of the counter might end up in geographically dispersed areas of the FPGA. The process by which the designer constrains the placement tool to use a set of predefined CLBs for certain parts of a design is called floorplanning [2]. The placement tool will start by mapping these parts first and will only have freedom for the rest. Floorplanning is an important technique to achieve timing goals in complicated systems such as those found in typical DSP applications.

PERFORMANCE BOOSTING TECHNIQUES

Besides working on the placement and routing of a design, a designer can also boost its performance by introducing architectural changes. Some of these changes are trivial enough to be delegated to the synthesis tool, for example the automatic insertion of extra internal buffers in a fan-out configuration to increase the current drive for a signal whose edges are being slowed by too many capacitive loads. If the synthesis tool detects that the output of a buffer is driving too many destinations, it will automatically replicate it. Registers, along with any combinatorial logic downstream, can be replicated in the same way and for the same reasons.

Another family of techniques consists in identifying places where the number of combinatorial logic layers between two register banks is too large. Figure 4 illustrates the technique of retiming, also known as “register balancing”.

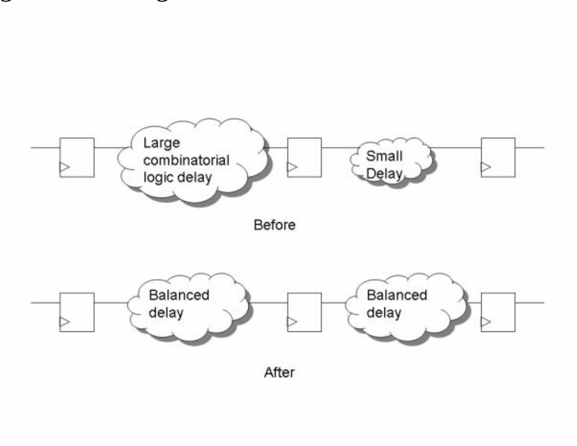

Figure 4. Register balancing.

If a design lends itself to this technique, then some of the logic after the first register bank can be displaced to the second register bank and the resulting system can be clocked at a higher rate. A variation of this theme, if a design can accommodate extra delay cycles from input to output, is to insert one or more register banks to break the large combinatorial block into smaller pieces. This technique is commonly known as pipelining.

Figure 5 illustrates a time-multiplexing scheme to double throughput.

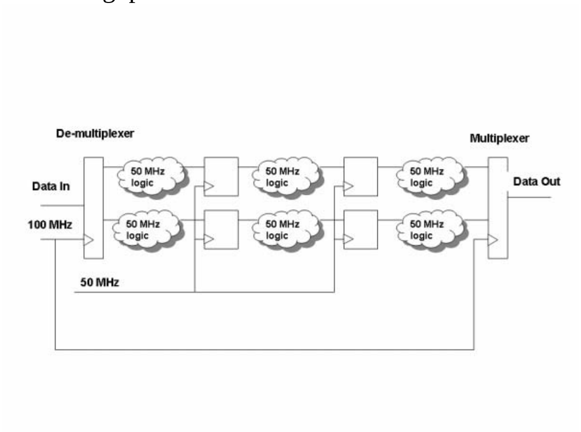

Figure 5. Doubling throughput through time multiplexing.

Time multiplexing and de-multiplexing can be used in conjunction with the other techniques described above to boost throughput by exploiting the inherently parallel nature of FPGA hardware. This is in fact a recurrent pattern in performance boosting schemes: a designer can often trade off speed (throughput) for silicon area. With silicon becoming cheaper and cheaper, there are fewer and fewer systems whose speed requirements cannot be met by FPGAs.

Another technique worth mentioning concerns multiplication of a variable signal by a fixed quantity. It is very often the case that an alternative arrangement can be found to avoid using a full-blown multiplier, saving silicon and increasing speed. As an example, one might want to calculate the product 0.5625M. But notice that 0.5625M = 9M/16 = M/2 + M/16, so a simple adder fed by bit-shifted versions of M will give the correct result. This technique is used for example in a lossy integrator filter that extracts the baseline from a Beam Position Monitor (BPM) signal in CERN’s PS.

DISTRIBUTED ARITHMETIC

Digital Signal Processing algorithms often involve the calculation of a Sum Of Products (SOP) of the following type:

$$y = \sum_{n=0}^{N-1} c[n] \cdot x[n]$$

Here the x[n]’s are N input data samples and the c[n]’s are N constant pre-defined coefficients, such as the coefficients of an FIR filter. If we assume x[n] has a width of B bits and we replace x[n] by its bit decomposition, we get:

$$y = \sum_{n=0}^{N-1} \left(c[n] \cdot \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \right)$$

After rearranging the sums (hence the term “Distributed Arithmetic [3]”), we get:

$$y = \sum_{b=0}^{B-1} 2^b \cdot \left(\sum_{n=0}^{N-1} c[n] \cdot x_b[n] \right)$$

Now there seems to be no major outcome of this manipulation until we realise that the term in brackets is easy to implement in a RAM or a LUT: for every possible set of x_b bits (consisting of N elements), we have to assign an output value. The resulting hardware structure is depicted in figure 6.

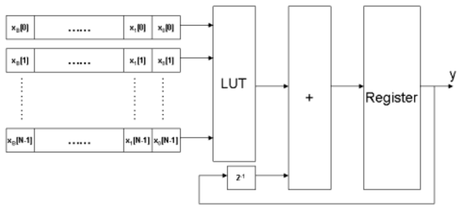
$x_B[0]$ ...... $x_1[0]$ $x_0[0]$
$x_B[1]$ ...... $x_1[1]$ $x_0[1]$
$x_B[N-1]$ ...... $x_1[N-1]$ $x_0[N-1]$
LUT
+
Register
y
2^{-1}

Figure 6. Distributed arithmetic implementation.

Input samples are all presented at the same time, one bit at a time, to the LUT. These N bits act in fact as address lines of the RAM or LUT. The data contained at a given address is the pre-calculated term in the parentheses of the equation above. This data gets scaled by ½ before going to an accumulator. After B iterations, the result of the computation is in the y register.

With distributed arithmetic we have achieved two important results:

- An SOP has been computed using no hardware multipliers. A pre-requisite for this is that the c[n] coefficients be constant.
- The number of cycles necessary to compute the SOP does not depend on the number of input samples but on the number of bits of these samples.

Of course, a larger number of input samples will require a larger RAM, but it will not create bigger delays, illustrating again the area for speed trade-off.

THE CORDIC ROTATOR

The basic arithmetic operations can be implemented in a fairly straight-forward way using digital gates:

- Addition of N bit numbers can be performed cascading N 1-bit full adders, each of them calculating a result bit and a carry to be fed to the next stage.
- Multiplication and division can be performed using a “pencil and paper” approach, i.e. scaling by powers of two and accumulating results under certain conditions.

For more involved operations, such as the conversion of complex numbers from polar to Cartesian coordinates or vice versa, or the calculation of trigonometric functions, the COordinate Rotation DIgital Computer (CORDIC) algorithm [4] provides an elegant silicon-efficient solution.

A CORDIC rotator block takes a vector (x, y) as an input and rotates it by a given angle to give the result (x’, y’) according to the transformation:

$$x' = x \cdot \cos\phi - y \cdot \sin\phi$$

$$y' = y \cdot \cos\phi + x \cdot \sin\phi$$

These equations are impossible to implement using basic gates. However, if we factor out the cosine term, we get:

$$x' = \cos\phi\left[x - y \cdot \tan\phi\right]$$

$$y' = \cos\phi\left[y + x \cdot \tan\phi\right]$$

Now, the trick is to proceed by angle jumps iteratively, accumulate these jumps and stop when the accumulator value is close enough to the target angle. If we constrain the angle jumps for iteration i to be such that:

$$\tan\phi = \pm 2^{-i}$$

Then, the multiplication by the tangent is a simple shift operation, and the only decision left is the direction of the angle jump for each iteration. The cosine terms are simply fixed multiplicative constants and can be applied at the end since their value does not depend on the jump direction:

$$\cos(\delta_i) = \cos(-\delta_i)$$

The final CORDIC equations are therefore:

$$x_{i+1} = K_i\left[x_i - y_i \cdot d_i \cdot 2^{-i}\right]$$

$$y_{i+1} = K_i\left[y_i + x_i \cdot d_i \cdot 2^{-i}\right]$$

With:

$$K_i = \cos\left(\arctan 2^{-i}\right) = 1 \Big/ \sqrt{1 + 2^{-2i}}$$

$$d_i = \pm 1$$

The CORDIC rotator can operate in two modes:

- In rotation mode, we give the required rotation angle as an argument and the CORDIC iterates until the target angle is reached, producing x and y as outputs.
- In vectoring mode, the CORDIC takes an input vector and rotates it until the vector aligns with the horizontal x axis, producing x and the rotation angle as results.

As an example, we can see how to calculate the sine of an angle with the CORDIC. To do this, feed (x=1, y=0) and the desired angle to the CORDIC in rotation mode. After the rotation, the y result is in fact the sine we looked for, since the magnitude of the vector is 1.

CORDIC algorithms generally produce one additional bit of accuracy for each iteration. A typical application with a 16 bit result running at 100 MHz would therefore calculate the sine of an angle in 160 ns.

AN EXAMPLE: RF CAVITY CONTROL

Let’s illustrate some of these techniques with a real example. The system described below [5] performs the control of the radio-frequency electric field inside a cavity of CERN’s Linac 3. Figure 7 shows a general view.

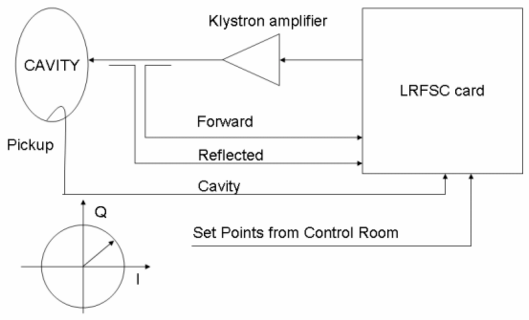

Figure 7. Block diagram of RF cavity control system.

At the fixed 100 MHz frequency imposed by the cavity, the RF signal can be completely characterized by two quantities: amplitude and phase (using polar coordinates in a phasor diagram) or I and Q if we work in Cartesian coordinates. The goal of the Low level RF Servo Control (LRFSC) card is to maintain a fixed value of I and Q from the cavity despite the imperfections of the klystron amplifier and disturbances due to beam loading.

The main feedback loop measures the signal coming from the cavity and performs the necessary corrections on the signal sent to the klystron. Forward and reflected signals from the amplifier output are also fed back to the card for cavity resonance control and compensation of large phase shifts in the klystron.

The card extracts a 20 MHz sine wave from the 100 MHz RF signal by down-converting with 80 MHz and low-pass filtering. The 20 MHz wave contains all the amplitude and phase information (or conversely I and Q) of the original 100 MHz wave, so we can extract these values from it and use them to determine the feedback. The 20 MHz sine wave is sampled at exactly 80 Ms/s and the samples are sent to an FPGA which implements the loop as well as VME communication and diagnostics.

The reason to sample at exactly four times the input frequency is that the output stream can then be interpreted as I, Q, -I, -Q, I, Q,… So with a trivial de-multiplexer and sign reversal scheme we can extract two 40 Ms/s data steams from the incoming 80 Ms/s stream, as shown in Figure 8.

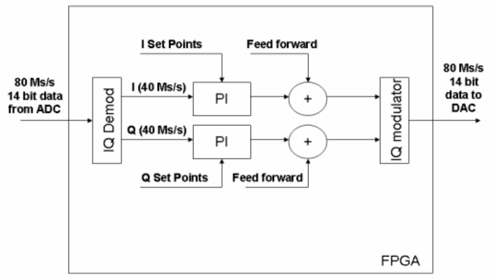

Figure 8. A simplified view of the LRFSC FPGA design.

The I and Q extraction scheme is straight-forward to implement and avoids the need for imperfect analogue components such as power and phase detectors. Once the I and Q streams have been separated, each goes through an independent Proportional-Integral (PI) controller implemented using the hard-wired multipliers of the Xilinx Virtex II family. There are also provisions for injecting systematic feed-forward corrections downstream before the signals go to the IQ Modulator block, where both streams are merged again to generate the samples of a 20 MHz output sine wave with the corrected I and Q values. These are sent to a fast DAC, the output of which goes through a low-pass reconstruction filter before being up-converted with the same 80 MHz local oscillator as the input. Finally the signal is band-pass filtered and amplified before being sent to the klystron, thus closing the loop.

The IQ Modulator/Demodulator pair plays the role of the multiplexer/de-multiplexer pair of figure 5, enabling us to have an 80 Ms/s system by splitting into two 40 Ms/s branches.

A future system enhancement will consist in adding an auxiliary loop to compensate for large phase shifts in the klystron. To implement the phase compensation, the phase will have to be extracted from I and Q, very likely using the CORDIC scheme described earlier.

SUMMARY

After briefly describing the internals of FPGAs and the associated design flow, we have given some tips and techniques commonly used to maximize the performance of a digital design in terms of clock speed. Two important FPGA techniques for performing arithmetic, namely DA and CORDIC have been explained and illustrated, and a real design example of Digital Signal Processing using an FPGA has been presented. The choice of topics has been rather arbitrary, but hopefully enough to give a flavour of the main issues concerning high speed DSP using programmable logic and to trigger design ideas in the field of Beam Instrumentation.

ACKNOWLEDGEMENTS

Many thanks to Wolfgang Heinze, Uli Raich and Tony Rohlev for fruitful discussions and to Bertrand Frammery for letting me set some time aside to delve into this interesting subject.

REFERENCES

[1] http://www.xilinx.com
[2] A basic introduction to floorplanning is available at http://www.fliptronics.com/floorplanning1.html
[3] U. Meyer-Baese, “Digital Signal Processing with Field Programmable Gate Arrays”, 2nd edition, Springer 2004.
[4] See http://www.andraka.com/files/crdcsrvy.pdf for an excellent survey of CORDIC algorithms for FPGAs.
[5] A. Rohlev et al. “All Digital IQ Servo-System for CERN Linacs”, EPAC 2004, Luzern, Switzerland.