design SuperCon 96

Building a High Performance Bit Serial Processor in an FPGA

Raymond J. Andraka

Andraka Consulting Group
16 Arcadia Drive
North Kingstown, RI 02852-1666 USA

1996
On-Chip System
Design Conference

the Andraka Consulting Group
16 Arcadia Drive • North Kingstown, RI 02852-1666 • USA

design SuperCon 96

Abstract

This paper describes the advantages and pitfalls of a bit serial architecture by studying the design of a vector magnitude processor inside a radar signal processor. The design combines bit serial arithmetic with a CORDIC algorithm to process 8 million 12 bit vectors per second inside a single FPGA. The bit serial architecture has the advantage of a very compact design solution that avoids many of the place and route problems commonly associated with FPGAs. The bit clock required to obtain the required data rate pushes the upper limits of today's FPGAs. Therefore, this paper also showcases the high performance FPGA design techniques needed to make a bit serial design more attractive. Finally, the paper discusses how to extend the techniques to develop any bit serial processor.

Authors/Speakers

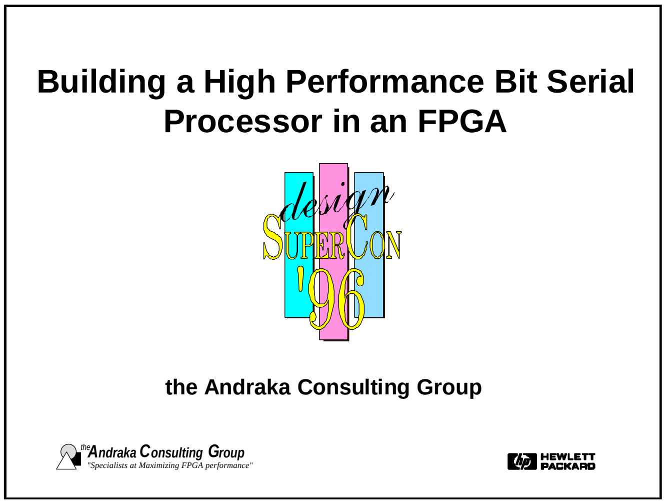
Raymond J. Andraka

Current Activities

Ray Andraka is the chairman of the Andraka Consulting Group, a digital hardware design firm specializing in high performance FPGA designs. His current consulting activities include supporting reconfigurable computer research, applying FPGAs to next generation general aviation avionics, and developing the electronics package for a new medical device. Ray is also writing application notes describing design techniques for FPGAs.

Author Background

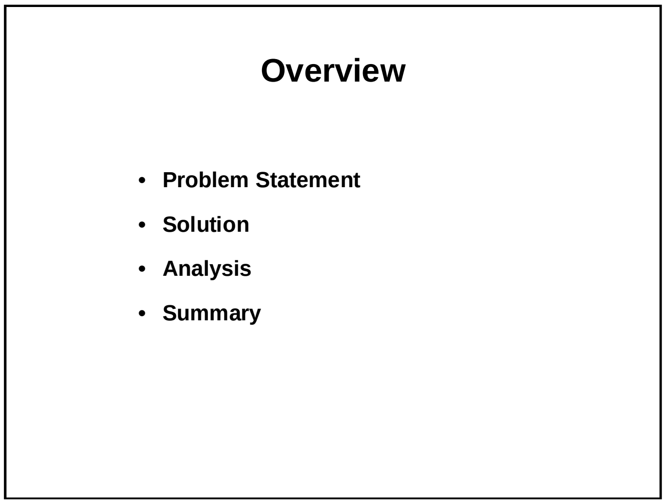
Ray Andraka has an MSEE from University of Massachusetts at Lowell and a BSEE from Lehigh University. He has originated and improved dozens of designs in Xilinx, AMD, Altera, Actel, Atmel, QuickLogic, and NSC FPGAs over the past 8 years. Many of these designs were for high performance signal processing applications. His signal processing experience includes over 5 years designing pipelined radar signal processors for Raytheon and 3 years of signal detection and reconstruction algorithm development for the US Air Force. He also spent 2 years developing image readers and processors for G-Tech. He presented a paper at the 1993 EE-Times PLD Conference describing his design of a 27 tap 12 bit FIR filter in an FPGA. That paper is the basis for designs in at least two FPGA vendor's macro libraries



Building a High Performance Bit Serial Processor in an FPGA

the Andraka Consulting Group

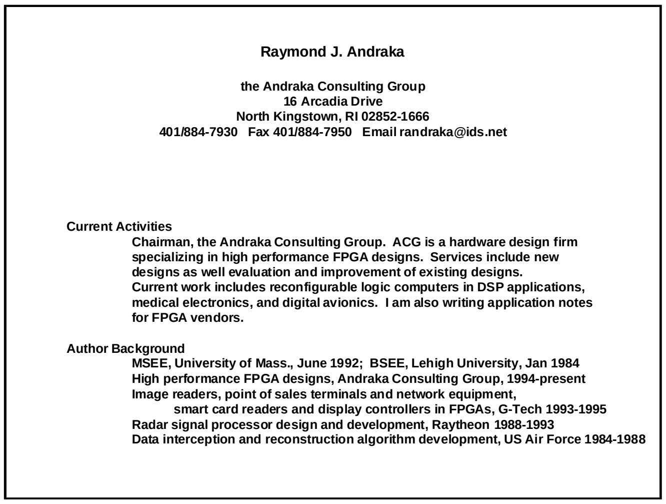
the Andraka Consulting Group
"Specialists at Maximizing FPGA performance"



Overview

- Problem Statement
- Solution
- Analysis
- Summary

In years gone by, hardware was expensive. A common design technique to reduce the costly hardware was to process data one bit at a time, reusing the same hardware for each bit. Today the cost per gate is a fraction of a percent of what it was then. With cheap hardware, the parallel designs have become so commonplace that bit serial solutions are often overlooked for applications where they may be the best choice. This is especially true when using an FPGA in signal processing applications. In those cases the limited size and routing capacity of the devices is pushed to its limit, often yielding unsatisfactory results.



Raymond J. Andraka

the Andraka Consulting Group
16 Arcadia Drive
North Kingstown, RI 02852-1666
401/884-7930 Fax 401/884-7950 Email randraka@ids.net

Current Activities
Chairman, the Andraka Consulting Group. ACG is a hardware design firm specializing in high performance FPGA designs. Services include new designs as well evaluation and improvement of existing designs.
Current work includes reconfigurable logic computers in DSP applications, medical electronics, and digital avionics. I am also writing application notes for FPGA vendors.

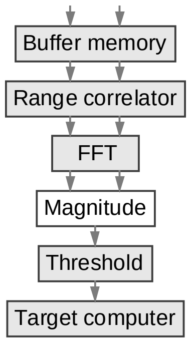
Author Background
MSEE, University of Mass., June 1992; BSEE, Lehigh University, Jan 1984
High performance FPGA designs, Andraka Consulting Group, 1994-present
Image readers, point of sales terminals and network equipment,
smart card readers and display controllers in FPGAs, G-Tech 1993-1995
Radar signal processor design and development, Raytheon 1988-1993
Data interception and reconstruction algorithm development, US Air Force 1984-1988

A bit serial approach to the design of these processors can alleviate many of the problems associated with FPGAs, including the headaches caused by the slow and limited routing resources. This paper presents the design tips and techniques to successfully build a high performance bit serial processor in an FPGA, beginning with the logic behind choosing a bit serial approach. In order to highlight the possibilities, the real world design of a radar vector magnitude processor is used to provide examples of the techniques discussed.

To better understand the problem, it is necessary to first review the function of the vector magnitude processor and the constraints on the design. In doing that, we will select and develop the most suitable algorithm. Once the algorithm has been defined, we will discuss how to efficiently implement it in hardware.

CW Radar Needs Vector Magnitude

- **12 Bit complex input**
- **12 Bit accuracy**
- **8 Mhz data**
- **95% Fault detection**



Most of the processing in the radar signal processor is performed on complex data in cartesian form. The target computer at the end of the process requires the data in polar form. A vector magnitude processor is used to convert from cartesian space to polar form. The FFT simultaneously outputs two 12 bit words representing the in-phase (I) and quadrature (Q) data every 125 nS. The target computer uses the vector magnitude and the range-doppler position of each point in the FFT output to determine presence, size, distance, direction and speed of the target. To minimize process noise, the vector magnitude has to be accurate to 12 bits. A hardware threshold function is added between the magnitude and the target computer to discard data with magnitudes below a programmable floor. This significantly lightens the process load of the target computer.

.

The Problem

- **Tight board space - single chip**
- **Low volume - no ASIC**
- **DSP microprocessor not adequate**
- **Parallel solutions won't fit FPGA**

The radar processor needs to fit in a fairly tight space, and the power budget is small. For these reasons, the vector magnitude processor, which is only a small part of the overall process, needs to be as small as practical. The board space allocated to the processor is roughly 6 square inches. This mandates a compact solution, and a single chip solution is preferred.

The total production run is only around 120 units including spares. A solution using an ASIC is therefore only acceptable as a last resort, as the non-recurring engineering costs do not get spread out over enough units to bring the cost down to a reasonable level.

The performance of general purpose DSP microprocessors is not adequate in this application. Even with advances in DSP processor design such as on chip cache, RISC code, out of order execution, and branch prediction, these processors still suffer from the inherent serial nature of their instruction streams. Computationally intense algorithms like the vector magnitude require many instructions per data point because there is no dedicated hardware to perform the function. Vector magitude requires at least 20 instructions per point to compute. Keeping up with the 8 MHz data stream would require a 6 ns instruction cycle.

The logic for a conventional parallel solution to the vector magnitude problem only fits in the largest FPGAs. The routing complexity for such a design exceeds the capability of most of those parts.

Solution Process

- **Define, test & decompose algorithm**
- **Hardware implementation**
  - **– Bit serial appropriate?**
  - **– design & optimize data path**
  - **– design control path**
- **Evaluate design**

As with any problem, finding a solution begins with understanding the problem. We are already aware of the performance and size constraints. In order to evaluate the options we also need to know what the processor has to do. We will first evaluate various methods of computing the magnitude, then select the algorithm best suited to a hardware implementation. The evaluation includes modeling the favored algorithm to verify suitability and to obtain test vectors for the hardware. Once the algorithm is selected, it is decomposed to provide insight into how best to implement it in hardware. After ascertaining that a bit serial approach is reasonable, the hardware detail design is done beginning with the data path. Finally, the completed design is evaluated for function and timing.

Alternative Algorithms

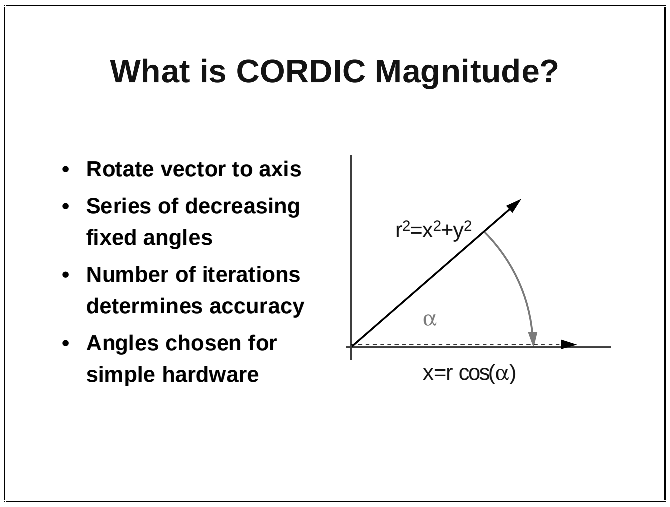
- **SQRT(X^2 + Y^2)**
- **Larger + 1/2 smaller**
- **Look-up table**
- **Specialty chips**
- **CORDIC**

The most obvious method of computing a vector magnitude is by using the familiar Pythagoras algorithm: r=Sqrt(x^2 + y^2). The square root function is very difficult to implement in hardware, and the two squares each require complexity approaching that of a multiplier. The Pythagoras approach is not very promising for a compact hardware solution.

A well known approximation of vector magnitude is the sum of the larger vector component and half of the smaller component. Later on, you may recognize that this is essentially the same as doing the first two iterations of the CORDIC algorithm. From an error analysis performed on the CORDIC algortihm, this only yields 3 bits of accuracy in the result. Our application requires a full 12 bits of accuracy. This approach is clearly not adequate.

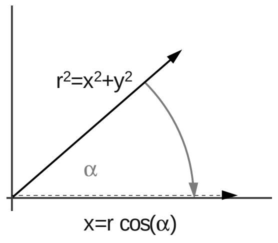
For smaller data widths, a lookup table can be used to convert the input data to a magnitude. In this case, the input is 24 bits wide (2 12 bit signed components) and the output needs to have 12 bits. Using an input sign reduction, the input can be easily reduced to 22 bits. The look up ROM needs to be 2^{22} (4M) words deep by 12 bits wide. Even with input sign reduction, this is a very large lookup table, and would require a board full of ROMs to implement.

There are a few specialty chips such as the TRW TMC2330 and the GEC Plessey PDSP16330 designed specifically for polar conversions. While these chips are available, they are typically very expensive. All of these parts are low volume products with single sources of supply. In the final analysis, the selected solution provides more function (the threshold function and much of the entire processor’s timing logic was pulled into the magnitude processor design) at a substantially lower cost.



What is CORDIC Magnitude?

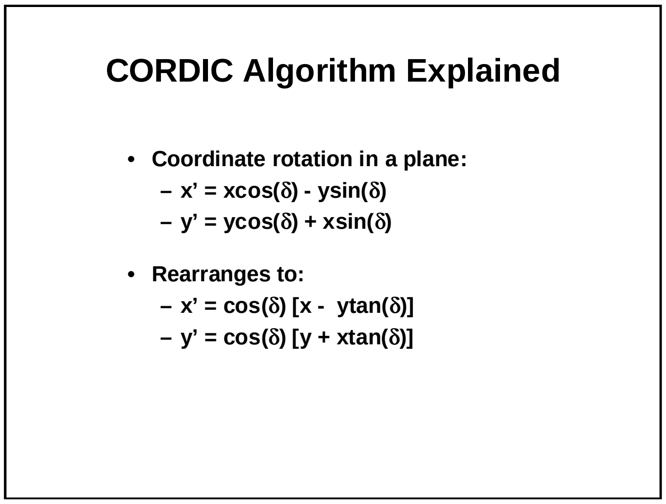
- **Rotate vector to axis**
- **Series of decreasing fixed angles**
- **Number of iterations determines accuracy**
- **Angles chosen for simple hardware**



CORDIC (COordinate Rotation DIgital Computer) algorithms are a class of iterative algorithms using only shifts and adds to realize the result. The specific algorithm for computing vector magnitude was the first CORDIC algorithm, and was the result of work in the late 1950's by Jack Volder[1]. Later work extended the CORDIC class to cover computation of many of the transcendental functions including all of the trignometric functions, the hyperbolic trignometric functions, exponentials and the inverses of all of these. The same architecture can also be used, although somewhat inefficiently, to perform multiplication and division. I also demonstrated a square root CORDIC algorithm in 1991 as part of my graduate work. The chief advantage of the CORDIC algorithm is that the hardware is relatively simple: Vector magnitude hardware is about the same complexity as an array multiplier!

The trignometric CORDIC functions all work by rotating a vector through an angle. For magnitude, the vector is rotated to the x axis, and the resulting x component is interpreted as the magnitude. Each iteration of the algorithm uses a successively smaller angle. The higher the number of iterations, the closer the rotated vector gets to the target angle, and therefore the closer the result is to the true magnitude.

The key to the CORDIC algorithm is that the discrete rotation angles are carefully selected to allow implementation using only shifts an adds.



CORDIC Algorithm Explained

- **Coordinate rotation in a plane:**
  - **$x' = x\cos(\delta) - y\sin(\delta)$**
  - **$y' = y\cos(\delta) + x\sin(\delta)$**
- **Rearranges to:**
  - **$x' = \cos(\delta)\,[x - y\tan(\delta)]$**
  - **$y' = \cos(\delta)\,[y + x\tan(\delta)]$**

The rotation of a vector in a plane through an arbitrary angle, δ, is mathematically described by:

$$x' = x\cos(\delta) - y\sin(\delta)$$

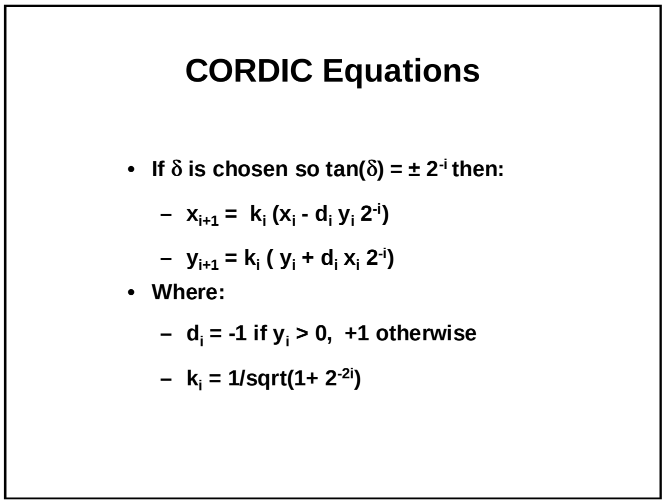
and $$y' = y\cos(\delta) + x\sin(\delta).$$

Rearranging these yields:

$$x' = \cos(\delta)\,[x - y\tan(\delta)]$$

and $$y' = \cos(\delta)\,[y - x\tan(\delta)].$$

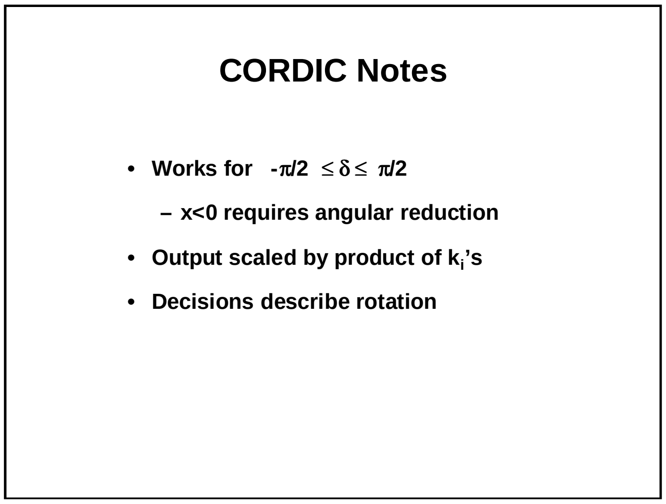
If we always rotate the vector by an angle of $\pm\delta$, the $\cos(\delta)$ term becomes a constant regardless of the rotation decision. That constant can be treated as a scaling constant at the end of the computation. If the rotation angle is further restricted so that $\tan(\delta) = \pm 2^{-i}$, then the rotation can be implemented using only a shift and an add (or subtract). By executing a series of successively smaller rotations, any arbitrary rotation angle can be achieved. The cosine terms of the successive rotations accummulate as a constant that is dependent only upon the number of iterations performed. The constant at each iteration is $\cos(\mathrm{atan}(2^{-i}))$, which reduces to $1/\mathrm{sqrt}(1+2^{-2i})$. The aggregate constant is the product of the constants from each iteration, which approaches 0.607253 as the number of iterations goes to infinity. In the case of the radar, the constant is simply factored into an aggregate processing gain attributed to the entire signal processor.



CORDIC Equations

- If δ is chosen so $\tan(\delta) = \pm 2^{-i}$ then:
  - $x_{i+1} = k_i (x_i - d_i y_i 2^{-i})$
  - $y_{i+1} = k_i (y_i + d_i x_i 2^{-i})$
- Where:
  - $d_i = -1$ if $y_i > 0$, $+1$ otherwise
  - $k_i = 1/\mathrm{sqrt}(1+ 2^{-2i})$

The decision function, d_i, is used to make the vector converge on the x axis. If the angle before rotation is positive (y is positive), a clockwise rotation is made, otherwise the rotation is counterclockwise. The sign of y before each iteration is easy to obtain, and works well to control the progress of the process.



CORDIC Notes

- Works for $-\pi/2 \le \delta \le \pi/2$
  - $x<0$ requires angular reduction
- Output scaled by product of k_i's
- Decisions describe rotation

The CORDIC magnitude algorithm will work for any angle with a magnitude less than the sum of the angles subtended at each iteration. That sum is approximately 100 degrees for 3 or more iterations. For convenience, we limit the input to the CORDIC processor to ±90 degrees. Vectors in the Left half plane (negative x) are easily accommodated by a simple angular reduction: negate the x and y components if the x input is negative, effectively doing a 180 degree rotation.. This reduction is included in my hardware.

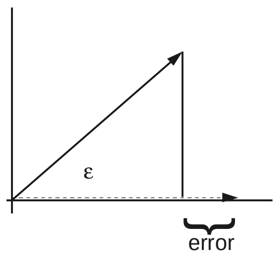
The CORDIC processor has a constant processing gain attributed to the constant cosine terms. This results in the output being scaled by approximately 1.647. In the case of the radar, this is lumped with several other fixed gains in the signal processor and dealt with in the post processor. Most applications can treat the gain in the same manner. In applications where the unscaled magnitude is needed, a multiplier is required to correct the scale.

The direction of each partial rotation is determined by a binary decision function, and the angle of rotation is fixed for a given iteration. It is therefore possible to determine the total angular displacement solely from the sequence of decisions made in computing the magnitude. A small ROM can be used to convert the decision vector to an angular displacement in any angular measurement system. Of course the components of the decision vector need to be aligned in time using appropriate delays to create the ROM address. Alternatively, an additional adder/subtractor can be added to the processor which either adds or subtracts the appropriate fixed angle at each iteration to accumulate the total angle in whatever units are convenient. The decision vector itself comprises an angular unit system referred to as a Binary Angular Measures or BAMs.

The fact that the decision vector describes the angle of rotation hints that a vector could be rotated through an arbitrary angle by controlling the rotation with the desired angle instead of the decision variable used for magnitude. Obviously, that angle has to be expressed in BAMs. This can be accomplished by either using a small ROM or by using an additional adder/subtractor as discussed above. In addition to general vector rotation, such a processor could be used to compute the sine and cosine of an angle (a unit vector placed on the x axis rotated through the desired angle will yield the sine and cosine). The processor gain can even be compensated for by scaling the initial unit vector by the reciprocal of the processor gain! Similar strategies can be used to arrive at the other trignometric functions and their inverses. This is exactly the mechanism used by early 'scientific' calculators to compute the trignometric functions.

Angular Error Analysis

- **Max err(α) = δ_n**
- **Err(r) = r(1-cos(ε))**
- **Accuracy improves by 2 bits/iteration**
- **7 iterations needed**



Truncation Error Analysis

- **Limited precision causes error**
- **Right-shifted data truncated**
- **3 extra LSBs needed**
- **Round final result to drop LSBs**

The output from the magnitude processor is the projection of the rotated vector onto the x axis. When the rotated vector lies exactly on the x axis, there is no error due to the angle. For most angles, however, the rotated vector angle will not be exactly zero. In that case, the computed output is reduced by the cosine of the angular error. The worst case angular error occurs when the next to last iteration produces a vector with a zero angle. The last iteration then moves the vector away from the axis by the angle of the last rotation. The magnitude result is bounded by the true magnitude and the cosine of the last rotation angle times the true magnitude. Note the error is biased; it always reduces the result. The desired accuracy determines the number of iterations required. The tabular analysis of the angular error shows a 2 bit per interation improvement in the accuracy of the result.

			maximum errors				
I	2^(-i)	rot angle	% mag	phase err	mag error	mag	scale
		atan(2^-i)	1-cos(a)	(deg)	% * full scale	bits	factor
0	1.000	0.785398	29.2893	45	1199.691	11	0.707107
1	0.500	0.463648	10.5573	26.56505	432.4262	9	0.632456
2	0.250	0.244979	2.9857	14.03624	122.2963	7	0.613572
3	0.125	0.124355	0.7722	7.125016	31.62982	5	0.608834
4	0.063	0.062419	0.1947	3.576334	7.976639	3	0.607648
5	0.031	0.03124	0.0488	1.789911	1.998536	1	0.607352
6	0.016	0.015624	0.0122	0.895174	0.499908	-1	0.607278
7	0.008	0.007812	0.0031	0.447614	0.124994	-3	0.607259
8	0.004	0.003906	0.0008	0.223811	0.03125	-5	0.607254
9	0.002	0.001953	0.0002	0.111906	0.007812	-7	0.607253
10	0.001	0.000977	0.0000	0.055953	0.001953	-9	0.607253
11	0.000	0.000488	0.0000	0.027976	0.000488	-11	0.607253

The result of the angular error analysis assumes infinite precision in the arithmetic. Further errors are caused by the truncation of the LSBs in the right shifted terms at each iteration. An approximation of the maximum error to be expected due to truncation can be found by summing the errors at each iteration when all the truncated bits are ‘1’s. The tabulated results below, show that for 7 iterations, the error can be as large as 3 least significant bits. To reclaim the accuracy, the processor needs to carry three additional bits of precision. The input data should be padded on the right with three zero digits to keep from losing the lsb contributions. The output can be taken from bit 3 and up to maintain the 12 bit path outside of the processor. To retain 1/2 lsb accuracy, a number equal to the bit 2 weight should be added to the result to round the result rather before truncating it.

i	2^(-i)	truncation	accumulated	bits
		error	truncation error	
0	1.000	0.000	0.000	0
1	0.500	0.500	0.500	1
2	0.250	0.750	1.250	1
3	0.125	0.875	2.125	2
4	0.063	0.938	3.063	2
5	0.031	0.969	4.031	3
6	0.016	0.984	5.016	3
7	0.008	0.992	6.008	3
8	0.004	0.996	7.004	3
9	0.002	0.998	8.002	4
10	0.001	0.999	9.001	4
11	0.000	1.000	10.000	4

Overflow Error Analysis

- Avoid internal overflows
- Extra MSBs allow growth
- Need one extra MSB
- Drop MSB in result

The processor described in this paper is a fixed precision integer arithmetic processor. As such, it is subject to overflows. While the effects of an overflow are predictable, the output can be very confusing and can be devastating to downstream processing. Overflows can be treated by using saturating arithmetic to minimize the ill effects. Overflows can be avoided entirely by adding sufficient 'guard' bits above the most significant bit to allow enough headroom to accommodate the maximum possible growth. The result can be truncated (or rounded) to retain the desired word width. Saturating arithmetic requires a considerable amount of hardware to realize and is not a perfect solution. In parallel systems, additional precision often requires more hardware than saturating arithmetic, so that approach becomes attractive. In contrast, the precision in bit serial systems can extended simply by increasing the length of the input word (may require extra delay registers). The only real penalty is the increased number of clocks required to process each word.

If both vector component inputs are allowed to go to full scale simultaneously, the magnitude of the resultant vector is 1.414 times full scale. The CORDIC process has a gain of approximately 1.6, which when combined with the maximum vector magnitude yields a maximum output of 2.33 times the full scale input. The data grows in the CORDIC process monotonically, so the maximum level is at the output. Without further knowledge of the input, the processor would require 2 'guard' bits to accommodate the maximum growth without overflowing. In our system, the combined effect of the limiting in the range correlator and the scaling performed in the FFT and correlator limit the magnitude of the input vector sufficiently to require only one guard bit.

The processor works on two's complement data, but by definition, the output is always positive. The guard bit in the output is the sign of the output, and is therefore always zero. Provided the target computer is expecting unsigned data, the extra bit carried through the process can be dropped at the output with no consequence.

Analysis of the algortihm shows that the X data path is always positive after the input angle reduction, so it could use unsigned arithmetic as long as a zero sign bit is provided to the Y path elements. The Y component needs to retain sign, but does reduce in magnitude as the algorithm progresses, so the number of bits in Y could be reduced after the initial rotation. In a parallel system these nuances can save a significant amount of hardware. A serial system, however, is much easier to deal with and uses less hardware if it has a fixed precision throughout.

Analysis Summary

- 7 Iterations
- 16 Bit internal precision
  - 3 Extra LSBs
  - 1 Extra MSB
- Round result to eliminate LSBs
- Drop result sign bit

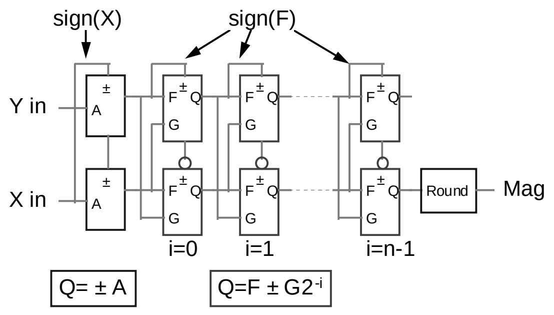
The various error analyses dictate the form of the CORDIC processor we need. The desired accuracy requires 7 iterations, which translates into 7 cordic stages. Errors caused by truncation require 16 bit precision internally to maintain the accuracy. The sign bit can be dropped at the output since the output is always positive, and the three extra LSBs can be eliminated by rounding to obtain the desired 12 bit output.

Model the Algorithm

- **C implementation**
- **Word size should match hardware**
- **Proof of algorithm and accuracy**
- **Provides test vectors for hardware**

For all but the most simple algorithms, I highly recommend modeling the algorithm in a high level language such as C before proceeding with the hardware design. This is especially true for iterative algorithms like the CORDIC since a seemingly small error can lead to drastically different results. The model should accurately reflect the precision and process through out the computation. Therefore, in the case of the CORDIC, it should model each iteration. The important result from the modelling is to observe any ill effects from truncation, overflow and other boundary conditions, as well as to verify the basic function of the algorithm. This will allow you to correct algorithmic problems before investing time in detailed hardware design. A very favorable benefit of modeling the algorithm, is that it provides a source of test vectors and intermediate results for tracing problems while simulating and debugging the hardware.

CORDIC Process



Mapping out the process provides clues as to how to implement the algorithm in hardware. The iterative process is drawn as a process flow, which is easily translated to a straight-through or pipelined design.

For applications where required performance is low, the algorithm can actually be performed by looping the data through the same hardware several times. The early 'scientific' calculators implement CORDIC algorithms using hardware loops and bit serial techniques to get away with an amazingly small amount of hardware. In our application, the desired throughput demands a pipelined design.

The pipelined 7 iteration design requires 17 adder-subtractors with pipeline registers inserted at convenient points. Most of the adder subtractors need to process 16 bit data (the first in each path could be reduced to 12 bits, and each after that needs to increase precision by one bit to a maximum of 16 bits). Total pipeline latencies up to several milliseconds are acceptable, as delays on this order do not affect the operation of the radar.

The logic for a parallel design will fit in some of the larger FPGAs such as the larger Xilinx 4K series parts. Unfortunately, the design cannot be routed in most of those parts, as they do not have sufficient routing resources to provide both the straight and crossed interconnect between the CORDIC stages. The parallel design should route in a Xilinx XC4025 FPGA as long as the design is properly floorplanned and no other logic is included in the device (this results in less than 20% logic utilization). The maximum performance of the parallel design implemented in the Xilinx XC4025 is less than four times the performance of the basic bit serial design presented in this paper!

Bit Serial Approach
A Quick Primer

- Process data one bit at a time
- Large hardware savings
- Clock per bit instead of per word
- Time-hardware product improved

Today, most computing is done on a word at a time basis. These bit-parallel designs operate on the entire width of a data word simultaneously. A bit-serial design operates on the data one bit at a time, reusing the same hardware for each bit. While this is inefficient timewise, the hardware savings can be enormous.

The majority of the processes we do on a data stream involve passing intermediate results across the width of the data word (carries for example). The propagation delays associated with this perpendicular flow can be significant, and directly influence the overall performance of a word wide system. In serial systems, the intermediate result can generally be held in a register until the bit that needs it arrives. The clock to clock propagation delays in a serial system are usually a small fraction of the delays in an equivalent parallel system. This means the time-hardware product of the serial system will be significantly higher than a parallel system (less than n serial processors are required to match the performance of an n bit parallel processor). The signal routing in a serial design also tends to very localized. In technologies like FPGAs where the routing resources are limited, this provides a substantial advantage. The local routing also helps to diminish the delay associated with the interconnect in FPGAs.

Is Bit Serial Appropriate?

- Algorithm adaptable to bit serial?
- Data rate achievable?
- Bit clock available?
- Pipeline latency acceptable?

A few questions need to be addressed when considering a bit serial solution. First, the algorithm needs to be able to be expressed as a serial procedure. Most algorithms can be serialized, although some are considerably more difficult than others to realize. Decomposition of the algorithm and transitive analysis on the subfunctions can be helpful in mapping out a serial strategy.

A second consideration when contemplating a serial solution is whether the data rate is achievable in the target technology. Modern FPGAs can realistically achieve register to register times as low as 8 ns in useful serial designs. The bit rate is therefore limited to around 125 MHz.. The overall data rate is the bit rate divided by the number of bits in the stream, including any overhead bits used for control. Expect to add one bit interval for resetting serial elements before each word. If the resulting rate is too low and enough unused logic exists it may be possible to use two or more processors in a parallel interleave arrangement to boost the data rate.

If the device can achieve the required performance, a bit clock must be available to clock the processor. In systems where the high bit clock is not available, a phase lock loop arrangement for synthesizing the clock can be used. The high frequency bit clock needed for high performance serial systems demands careful attention to transmission line effects and electromagnetic interference.

Finally, the application for the processor must be able to tolerate any pipeline delay introduced by the serial processor. The latency in a parallel system is frequently as high or higher than the equivalent serial system, so this is rarely a concern.

Bit Serial Conversion

- **Decompose algorithm**
- **Use standard serial solutions**
- **Transitive Analysis for unusual functions**
- **Relative delays for bit shifting**

The conversion of a parallel design to a bit serial algorithm begins with separating the algorithm into its component parts. Once that is done, many of the more common functions can be directly implemented using readily available serial solutions. Solutions to the more unusual functions can be identified using transitive analysis.

Transitive analysis can be very valuable for understanding the vagaries of implementing unusual functions in serial logic. Transitive analysis is simply the practice of determining which output bits are affected by the values of each input bit. This not only indicates the preferred shift direction, but also indicates which bits need to be held for future computations and hints of the complexity of the function.

Since the data is presented a bit at a time, bit shifting is accomplished by delaying one bitstream relative to another. Most of the time, this will require some special treatment to handle the misaligned word boundaries. An example of the shifting is found in the magnitude processor presented in this paper.

Transitive Analysis

- **Left transitive functions - lsb first**
- **Right transitive functions - msb first**
- **Intransitive functions - trivial**
- **Full transitive functions - difficult**
- **Ripple transitive - preferred form**

The majority of arithmetic functions are left transitive, meaning only outputs of equal or higher order than (to the left of) the input bit are influenced by the value of the input. Left transitive functions therefore favor right shifting so that the LSBs are presented first. These functions include all of the unadic arithmetic operators, addition, subtraction, gray code and bcd conversions.

Right transitive functions have outputs that are affected only by the inputs of higher order than the output. The most notable example of a right transitive function is the compare. If your processor is primarily performing sorting or other compare intensive operations, MSB first may be the preferred shift direction. It is worth noting that compares can be done in a predominantly left transitive system using subtraction and detecting sign and overflow in the result. This requires more complexity than the msb first logic, but is easier than switching shift direction in the middle of a process.

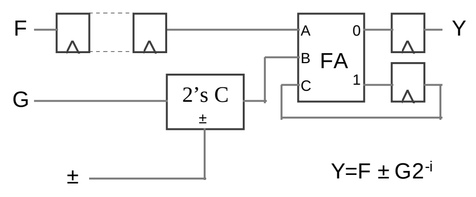
Intransitive functions are those where the output is only influenced by the inputs of the same order. The bitwise logical operators all fall in this class. These functions are trivial to implement in serial systems.

Functions that sample several or all of the input bits to produce an output not dependent on the input bit positions can be treated as either left or right transitive. Examples of these are tally adders, parity trees, and zero detection circuits.

Fully transitive functions are those which every input affects every output. I include partially transitive functions that cannot be classified as either right or left transitive in this class, since these get treated the same way. Functions in this class include bit rotations, bit field operations, and arbitrary encode/decode functions. These functions can be difficult to translate to bit serial mechanisms. Sometimes they are best handled in the parallel realm. Fortunately, most of the functions you will be dealing with can be classified as left or right transitive. A little ingenuity often helps find alternative solutions for those functions that are not. Translation of this difficult class is beyond the scope of this paper.

Ripple transitive functions with a distance of n are those whose output can be formed the from the current input and the n previous inputs and outputs. Previous in this case refers to either the next higher or next lower bit depending on the direction of the ripple transitivity. This class of functions is interesting because many of the left or right transitive functions can be expressed more compactly in a ripple transitive form. The unadic artihmetic operations, add and subtract, for instance, can all be expressed as left ripple transitive functions with a distance of 1. This means that each output can be expressed as a function of an output from the previous bit and the current input. Multiplication also can be expressed in a ripple transitive form, which is the basis of the serial by parallel multiplier. Since this form has the effect of simplifying the hardware, it is the preferred form. The ripple transitive form is often the intuitively obvious form of the function.

Shift-Add/Subtract Element

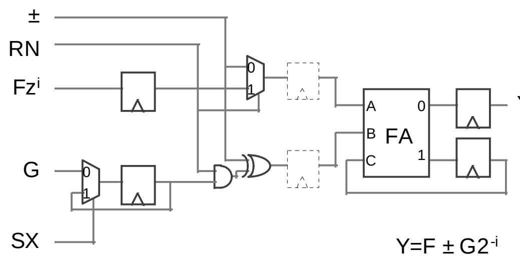


The development of the shift-add/subtract element used throughout the CORDIC processor is illustrative of the conversion process. Notice the carry from the full adder is registered (effectively a 1 bit shift) and returned to the carry input of the adder. This is an implementation of the ripple transitive form of the adder. It is a stock solution.

The 2^{-i} shift is accomplished using delays to realign the bits in F relative to the bits in G. The data is presented LSB first. In order to shift G i bits to the right, the bits in G must arrive at the processing element i bit times before the corresponding bit in F. By inserting the delay on the F path, the F data is shifted to the left relative to G which is the same as the G data being shifted to the right relative to F.

There are two problems with the element as shown here. First, , the carry flip-flops need to be cleared before each new data word so that the previous word's carry out is not added to the current word. There is a similar clearing concern in the 2's complement stage. The other problem is that when intentional bit misalignments are introduced, the word boundaries also become misaligned. This causes the dangling lsbs on the next right shifted word to be processed with the msbs of the other data stream(s). Extra logic needs to be added to eliminate the overlap between the words on shifted inputs. In most cases, the best way to do this is to replace the dangling lsbs on the right shifted word with either zeros for unsigned systems or the extended sign of the previous word for two's complement systems. Note also, that extra pipeline delays need to be added to the F path to match any delays in the 2's Complement stage so that the data retains it's intended alignment.

Improved Shift-Add/Subtract



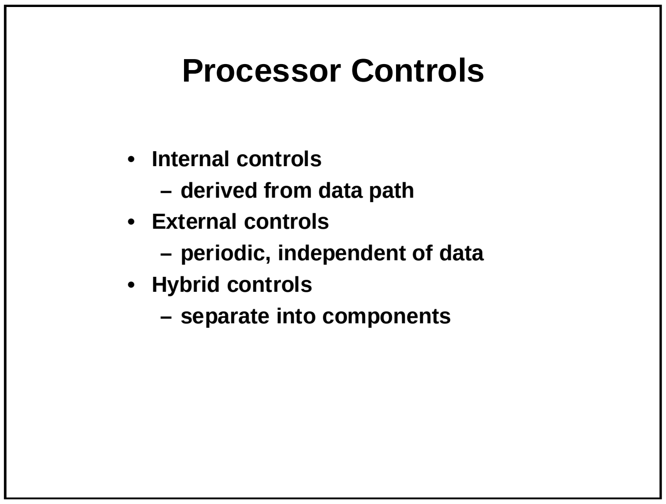
The function of the two's complement stage in the original design can be decomposed and partially combined with the adder that follows it. Two's complement is found by inverting the operand and adding one to the result. Setting the carry bit in the adder (essentially forcing a carry in) before the the data arrives causes the the sum to be increased by one. Therefore, we can invert the input to be subtracted and set the carry to realize subtraction. The familiar serial algorithm (pass the data until the first '1' bit then invert) is in fact just a logic reduction of a serial adder with one input at zero and the carry set before the operation begins. To create a selectable add/subtract, you simply need to control the inversion on the subtracted input and either set or reset the carry flip flop as appropriate before the operation begins. The inversion is easily controlled using an exclusive OR gate.

Synchronous design is essential to achieving the high performance required. The flip flop direct sets and resets are often not recognized as being asynchronous. These should not be used in the design (they can be connected as a global reset for power up initialization-but don't waste routing resources connecting them). The word cycle resets and sets should be done using added logic to realize synchronous behavior. These controls do not necessarily need to be introduced right at the targeted register. In our design, that could cause unnecessary routing congestion. The effect of a set or reset can be had by forcing the value of the data upstream of where the reset is needed. The carry output of the full adder (this is the register that needs to be reset) is the majority function of the adder inputs. Its value can be forced by controlling any two inputs to the adder. The logic used in this implementation forces 1's or 0's into the F and G inputs to set or reset the carry. The injection point for the reset is convenient to the controlled inversion as well.

The drawing shows ghosted flip-flops after the reset and twos complement logic. These registers are not functionally needed, but in many of the FPGA technologies they should be inserted to break up the otherwise long combinatorial delay. Even with parts that allow enough inputs and logic to implement the adder and reset logic in one cell, it may be advantageous for routing reasons to insert the register to force the use of two cells.

This improved shift-add/subtract element also solves the misaligned word problem introduced by the shift of G relative to F. The SX control passes G data until the sign bit arrives, then holds the value of the sign for the rest of the word, effectively extending the sign over the lsbs of the next data point . Sign extension is considerably more difficult to accomplish in an msb first system.

The delay queue on the F input still exists, as represented by the z^{-i} on the F label. I have not included the registers in this drawing for clarity, and to aid in the understanding of the control logic developed next.



Processor Controls

- **Internal controls**
  - **derived from data path**
- **External controls**
  - **periodic, independent of data**
- **Hybrid controls**
  - **separate into components**

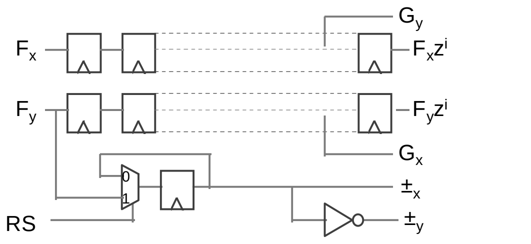
The serial data path normally includes a number of controls that influence the execution of the function. I classify these controls as internal and external controls.

The internal controls are those derived from the datastream, and are usually dependent on the intermediate values of the data. These controls are generated entirely from within the data path processor. Examples of internal controls are those derived from the sign of data or from the results of comparisons. Adding the internal controls often requires additional logic and pipeline delays to extract and align the control with the data. These adjustments to the data path can effect the timing of the external controls and may add additional external controls. This is why I treat the internal controls first.

The external controls are not affected by the values of the data stream. The external controls are generally periodic with a period equal to the word rate. These controls are created external to the data path by some form of sequencer. This class includes all the timing controls to the data path such as resets, sign extension and latch controls. The timing of the external controls is affected by added pipeline delays, so their design is best left until after the data path has been optimized for a particular FPGA.

Occasionally a control is a hybrid of the two classes. In these cases, it is usually advantageous to keep the internal and external components of the control separate until the point where they are used. This practice sometimes even allows the logic to be partitioned more efficiently.

Shift-Add/Subract Internal Control



F	R	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	S	R	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	S	R	0
A/S Ctl FF Q	S																R	S																R	S	
A/S FF																			R	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	S	R
sign ext ff Q																		R	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	S	R	0
Fz^{-1}																	R	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	S	R	0	1
L																	H																	H		

The only internal control in the shift-add/subtract element is the add/subtract control. The magnitude CORDIC algorithm uses the sign of the Y component from the previous iteration to select add or subtract (the control is inverted to one path so one adds when the other subtracts).

The sign bit is the last bit presented since the word is shifted lsb first. A pipeline delay is required to delay the data to the next stage until the sign is available. The length of the delay pipe depends on the word length and the relative delays between the sign pick-off and the control application.

The value of the sign affects the process of the entire data word, so the sign value has to be held. A simple register and multiplexer arrangement satisfies this, but requires an additional external control signal.

I have found that a timeline is a useful aid for ensuring the proper alignment of control and data. It reveals the delays necessary to keep data and control aligned, as well as nature of the control signals. I use a spreadsheet for the timeline, as it allows quick reproduction of the data stream and easy modifications

The delay used to shift the G data with respect to the F data can be absorbed into the delay required to obtain the sign. The G data is taken off a tap in the delay pipe an appropriate number of bits ahead of the F output. The tap is different for each stage in the pipeline. Pipeline reductions like this are frequently possible with the proper selection of control pick-offs. A matching delay is required on the X data path to keep the X and Y data aligned.

Device Selection

- **Registers and IO**
- **Speed**
- **Cell architecture - fine grain superior**
- **Storage technology - RAM**
- **Tools & availability**

Any FPGA is suited to bit serial design. The first consideration is whether there is sufficient logic and I/O resources to support the design. The minimum number of registers needed can be obtained from the groundwork we've already done. In most cases, the logic ahead of each flip flop is fairly minimal or can be broken into a couple of pipeline stages. Most bit serial designs have a very low routing complexity, so the routing resources are not an issue.

For high performance bit serial designs, the cell and routing delays play a very significant role in the success or failure of a design. Examination of the timing published for a one bit adder or a similar macro in a vendor's library will give a ballpark figure of the types of speeds you can expect to get from each device.

Finer grain architectures tend to be superior for bit serial designs because they offer a higher register to logic ratio. The serial designs tend to have many registers with little or no combinatorial logic in front of them, so much of the logic in coarse grained arrays goes unused. One caveat to using fine grained devices is that some seemingly simple logic functions may require several cells to implement, or may require some changes in logic to tailor the algorithm to the logic. Look-up table devices are the easiest to use, but beware of the register counts and resist the temptation to use all of the inputs to a cell.

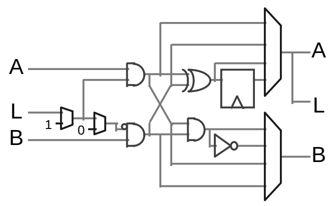
The RAM based FPGAs are a boon to in circuit testability. The fault isolation requirement for this processor is easily met by loading a test function into the FPGA in place of the vector magnitude. The drawbacks to RAM based parts are that they need to be loaded from an external device or bus before they do anything, and that the device programs are not secure from reverse engineering.

A final consideration in part selection is the price and availability of the part and the usefulness of the tools. The price and availability of the parts are superficial to this paper. For high performance work, the tools must, at a minimum include relatively painless floorplanning, an accurate static timing analyzer that operates on the routed design, and the ability to inspect and directly manipulate the placement and routing. A good macro generation facility that allows hand place and route is also high on the should-have list.

Based on these considerations, as well as some external influences we chose the National Semiconductor CLAy 31™ for the vector magnitude processor.

National Semiconductor CLAy 31

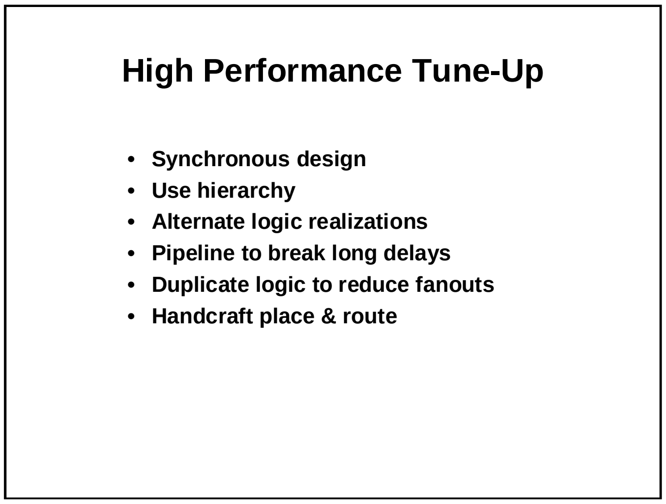
- **3136 registers**
- **108 I/O**
- **~2ns / cell**



The CLAy 31 cell is basically a half adder with a register on the sum output. It contains a little extra logic to allow configuration as a 2 input mux and a few other basic functions. Each cell has 2 direct connections via the A and B wires to each of the four nearest neighbors, and connections to horizontal and vertical local busses (L). This cell architecture is extremely well suited for bit serial processors.

A basic serial adder can be made using four cells, although a six cell variant is more routable. The six cell adder has a worst case clock to clock time of 11.5ns. Pipelining the serial adder reduces the minimum clock period to 5.8 ns.

One of the strengths of the CLAy 31 is the sheer number of registers it contains (3136 cells with a flip-flop per cell). I've done a 27 tap bit serial correlator [2] in this part, and it is possible to fit a serial FIR filter with over 50 taps.



High Performance Tune-Up

- Synchronous design
- Use hierarchy
- Alternate logic realizations
- Pipeline to break long delays
- Duplicate logic to reduce fanouts
- Handcraft place & route

The serial architecture derived so far is already well matched to nearly any FPGA. The biggest headaches with FPGAs are usually caused by routing complexity exceeding the ability of the resources. A serial design has minimal routing by nature. There are still several things which can be done to tune the design for better results.

First, as I mentioned earlier, the entire design should be synchronous, including the sets and resets. If you have followed my advice so far, this is already taken care of.

Use hierarchy in the design. This keeps the design organized, makes it easier to select macro boundaries, and speeds the design cycle. Some of the automatic place and route tools also look at the heirarchy for hints when placing the design.

Some logic may not map efficiently into the cell architecture in FPGAs that do not use look-up tables. The National Semiconductor device, for example does not implement OR gates very well (the OR gate in the library must connect one input to a local bus). Look for alternate logic realizations that will fit the architecture better.

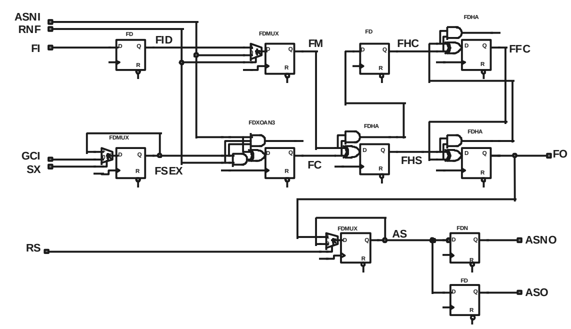
Add pipeline registers to break up long delays. Sometimes, rearranging the order signals get combined in combinatorial logic will permit better partitioning. When adding pipeline delays in a path, remember to add an equal number in any parallel paths, including control timing. A frequently missed source of long delays is in the routing. The clock to clock timing can often be improved by “pipelining the wires”. This is usually not obvious from the schematics.

The serial designs normally do not have very high fan-outs in the data path. Be aware though, that even small fan-outs (as low as three) can affect timing enough to ruin some of the fastest designs. Eliminate high fan-outs by duplicating the signals at the source or in a pipelined distribution tree. The external controls are especially susceptible to high fan out. On those controls, I prefer to pipeline the control, taking taps off the pipe at the correct phase closest to the destination. Doing this will usually reduce the master timing to a very simple ring counter.

On very high performance designs it is absolutely critical to understand how the logic fits into the cells and what interconnect is used. I strongly recommend careful floorplanning at this stage. A time saving technique is to design macros for the subfunctions. Place and route the macros as hard or relationally placed macros. In the design of such pieces, awareness of the locations of the I/O relative the adjacent functions goes a long way toward a good design. There are always those who will frown on the detail at this stage, but the human mind is still the best place and route tool there is and the layout is the most critical link to maximum performance.

Serial processors are generally compact enough that in cases where a single processor does not meet the required cycle time, additional processors can be used in a parallel interleave fashion. Those algorithms that operate on a block of data rather than on individual data points are generally not well suited for parallel interleave acceleration.

Finished Shift-Add/Subtract



The finished design of the shift-add/subtract element incorporates a number of the performance tune ups. None of these improvements would have been apparent without floorplaning.

The design shown has the clock and asynchronous resets omitted for clarity. The asynchronous resets are only for power-up reset. The processor is initialized synchronously using the external controls RNF, RNG (not shown) and RS at each stage.

The traditional OR gate in the carry logic of a full adder has been replaced with an exclusive-OR (the 1-1 input condition is impossible in this case) yielding a much faster and easier to route design. This is a case where the device architecture influences the design.

The single level design of the full adder required a minimum clock to clock time of 11.5 ns. Adding a pipeline register halves the minimum period to 5.8ns. The full adder inputs had to be swapped around to avoid putting the pipeline register in the carry feedback loop (the non-pipelined six cell design adds the fedback carry at the first half adder).

The inverted add-subtract control was originally obtained from the AS flip flop through an inverter. The long route combined with the multiple loads on the AS signal caused the delay to exceed the target value of 8ns. Adding registers, essentially pipelining the wires, brought those delays in line. The delay queue was extended to keep the data and control aligned.

I handcrafted a placed and routed macro containing a pair of the adder-subtractors and the a/s control. Each stage of the processor consists of one instance of this macro, the external control logic (a macro) and 2 delay queues. By doing the majority of the work in reusable macros, the entire design was completed inside two weeks.

External Controls

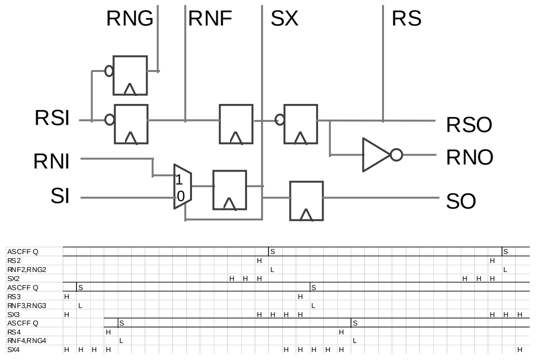
- **Use a Timeline**
- **Route controls perpendicular to data**
- **Keep fanouts low**
- **Generate control with pipeline**

When the data path logic is laid out, run the external controls on long lines or busses extending perpendicular to the general data flow. If the desired control signal connects to more than one column (assuming the data is flowing across), split it into separate controls. In cases where the control has multiple loads in the same column, it may be worthwhile splitting the load to reduce the delay due to fanout.

The design of the external control pipeline should wait until after the data path design has been completed, including layout. Again, a timeline showing the data flow through the processor is invaluable for deriving the controls. The external controls are periodic, and should be easy to generate. Use the timeline to find the timing relationships between every external control signal and the data. This will provide the phasing information you need to build the control pipeline.

Higher performance can be obtained from the design if the control is treated as a pipeline, where a stimulus is injected into the front and then ripples down the pipe providing the control as it propagates. This avoids long routing and allows the control to be distributed. The stimulus in this scheme can often be the output from a ring or LFSR counter.

External Control Pipe Section



The external control is provided by a pipeline. The top pipeline provides the reset signals corresponding to the reset bit time in the data. This gets phased properly to match the data everywhere it is needed. The bottom pipeline generates the sign extension control, which grows one bit time longer in each subsequent pipe section. This pipe section is repeated for each cordic stage. The first stage input is derived from the parallel to serial shift register load pulse. The three outputs on the right connect to the corresponding inputs of the next stage. The layout of the control pipe is designed to match the 4 cell width of the shift-add/subtract element and delay queues.

Design Verification

- **Functional Simulation - Viewlogic**
  - **No delays**
  - **Check proper Function**
- **Static timing - FPGA timing tool**
  - **Max register to register delays**
  - **Indicates maximum bit clock**

The first step in a design verification is functional simulation. Use Viewlogic's Viewsim or an equivalent logic simulator. Don't bother with modelling the delays, the goal here is to verify the the proper function. Use the same vectors used with the C model so that the model outputs can be compared with the simulation results. The simulation data set should minimumally cover all of the boundary conditions and all possible combinations of the signs of input vectors. In the case of the CORDIC processor, a minimal set would include vectors from all four quadrants with all combinations of maximum and zero magnitudes.

After the function is verified, the timing should be checked to make sure the specification is met. If you've done your homework in the design phase, you will already have a good feel for where the timing lies. Nevertheless, a thorough static analysis should be done to verify the system will run under all conditions. Most of the FPGA tools now include timing analyzers that provide worst case delays between various elements of the design. Be cautious of the timing reports on some of the tools; the tool set up can be complicated and it is easy to miss some paths. Also, make sure the tool is set up for worst case timing, not typical or best. A hierarchical design using prerouted macros saves a lot of effort in the timing analysis.

The Atmel tools indicate entire CORDIC processor has a worst case register to register delay of 8 ns in the -2 part. For the 17 bit data word (16 data bits plus the reset bit), this means a 136 ns word cycle time. A single processor does not meet the requirement. A CORDIC pipeline processor only occupies about 25 percent of the FPGA, including the I/O shift registers. In order to boost the apparent performance, I used two instances of the processor, interleaving the data so that the apparent minimum cycle is reduced to 68ns. While there are enough unused logic cells to allow up to 4 pipelines. The shape of the completed processor is such that only 2 parallel processors will fit. The design could probably be laid out differently so that at least 3 and maybe four pipes would fit with similar performance. This design was purposely done with tall narrow cells to leave space for post processing. If symetric sampling at the input and/or output is required for all phases, the precision of the processor can be increased to make the cycle time evenly divisable.

I am not a fan of timing simulation for synchronous designs. There are almost always timing situations which are either not set up properly and not detected or which are not recognized and therefore not simulated. A thorough static analysis will uncover all the potential timing problems.

Design Assessment

- **30 x 22 cells per processor pipe**
- **7.35 MHz data (14.7 MHz w/ 2 pipes)**
- **80% of FPGA left usable**

The CORDIC processor studied in this paper occupies a rectangular area 22 cells high and 30 cells wide, not including the output rounding and the I/O shift registers. Nearly 80% of the device is open for the I/O and other uses. The design uses a remarkably high 67% of the cells it covers for logic. Less than 2% of that logic is unregistered. The majority of the remaining cells under the macro are used for wiring.

A single instance of the pipeline can handle word data rates up to 7.35 MHz under worst case conditions. Adding a second pipeline in a parallel interleave arrangement doubles the maximum data rate to 14.7 Mhz, and uses a little over 42% of the device.

The pipeline latency of the CORDIC processor is 148 bit clock cycles. That translates to a latency of less than 9 word times. The equivalent parallel processor would need more than 9 pipeline delays to achieve performance any better than the serial processor.

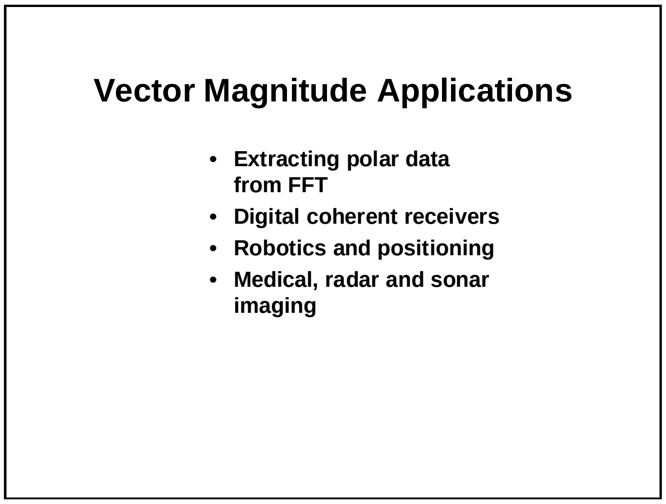
Parallel Comparison

- **Upper limit is about 20 Mhz**
- **Adders alone occupy 70% of device**
- **Not routable**
- **Minimum 15 clock latency**

An estimate of the size and performance of an equivalent parallel processor in the same device can be made by using the characteristics of a 16 bit adder. The best performing 16 bit adder in this device has a maximum clock frequency of 20.5 Mhz assuming the inputs are registered right at the adder. That adder fully occupies an 8 x 16 cell rectangular area. The 17 adders needed in this design alone cover 70% of the available area in the array. A compact 16 bit adder can be realized in half the area with a 2 x 32 cell footprint, but its performance limits the clock to a mere 13.2 Mhz assuming registered inputs and outputs.

The wiring of the cross terms in the processor requires an abundance of long routes. The routing resources of the FPGA are too limited to simultaneously provide the needed connections and the required logic density. The parallel realization of processor will not route in this part. Even if it did route, the performance is not much higher than can be done with a serial processor!

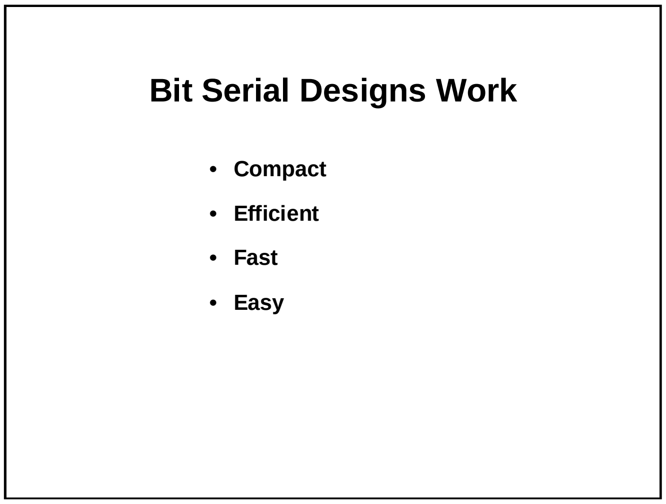
With special care and low logic utilization, the parallel design looks like it would be feasible in the largest Xilinx 4K parts. Even in those parts, however, the best performance would only be about 20MHz based on the 16 bit adder performance!



Vector Magnitude Applications

- Extracting polar data from FFT
- Digital coherent receivers
- Robotics and positioning
- Medical, radar and sonar imaging

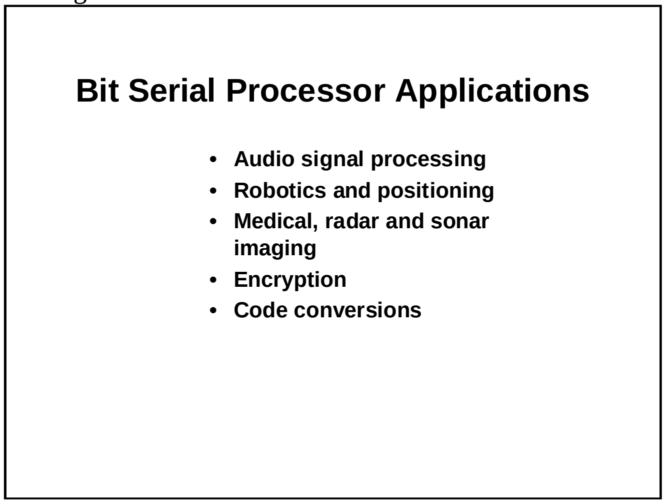
The CORDIC vector magnitude has application wherever a conversion from cartesian to polar space is required. These potential applications include FFT post processing, digital recievers, motion control and sensing in robotics, and most imaging applications. Minor modifications to the CORDIC processor presented here can provide trignometric functions of input angles, their inverses, hyperbolic trig functions and their inverses, exponentials and logarithms among other functions.



Bit Serial Designs Work

- Compact
- Efficient
- Fast
- Easy

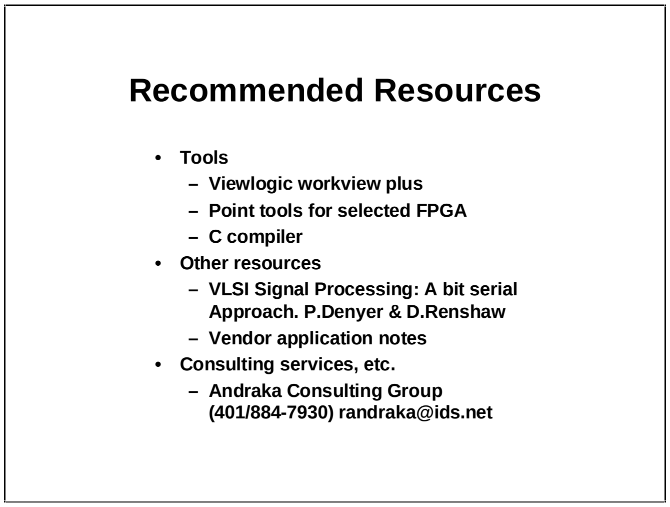
In this paper I used a real world example of a RADAR application to demonstrate the techiques used to create a high performance bit serial processor. The simple interconnect and compact logic allow these processors to significantly exceed the processing power per gate of equivalent parallel designs. High speed design techniques make bit serial a viable alternative even in moderately high performance systems!



Bit Serial Processor Applications

- Audio signal processing
- Robotics and positioning
- Medical, radar and sonar imaging
- Encryption
- Code conversions

Bit serial processing is applicable in nearly any system where the data rate is achievable by a serial system. The potential for enormous reductions in hardware makes bit serial design attractive anywhere more bang for the buck is needed. Prime applications are audio, motion control, imaging, encryption and code conversions.



Recommended Resources

- Tools
  - Viewlogic workview plus
  - Point tools for selected FPGA
  - C compiler
- Other resources
  - VLSI Signal Processing: A bit serial Approach. P.Denyer & D.Renshaw
  - Vendor application notes
- Consulting services, etc.
  - Andraka Consulting Group (401/884-7930) randraka@ids.net

[1] J.E. Volder, “Binary Computation Algortihms for Coordinate Rotation and Function Generation,” *Convair Report,* IAR-1 148, Aeroelectronics Group, June 1956.
[2] R.J. Andraka, “FIR Filter fits in an FPGA using a Bit Serial Approach,” Proceedings of the EE-Times 3rd Annual PLD design Conference and Exhibit, March 1993.
[3] C.R. Rupp, “Digital Function Analysis and Synthesis,” Course notes to 16.674, University of Massachussets at Lowell, 1991
[4] P. Denyer & D. Renshaw, “VLSI Signal Processing: A Bit Serial Approach. Selected Readings,” Addisson-Wesley, 1985.