Introduction to Field Programmable Gate Arrays

Lecture 2/3

CERN Accelerator School on Digital Signal Processing
Sigtuna, Sweden, 31 May – 9 June 2007
Javier Serrano, CERN AB-CO-HT

Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

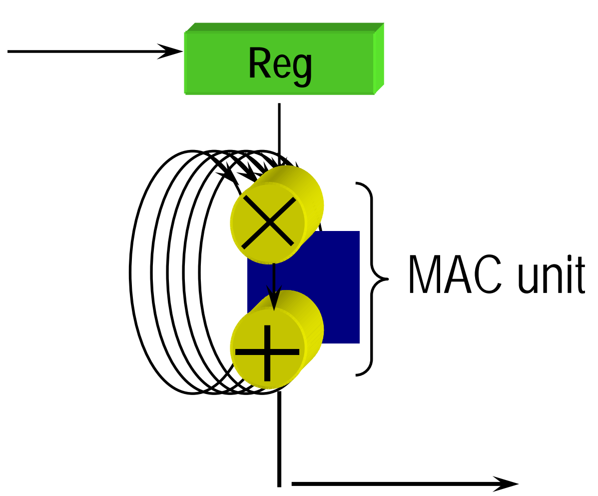
Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

Why FPGAs for DSP? (1)

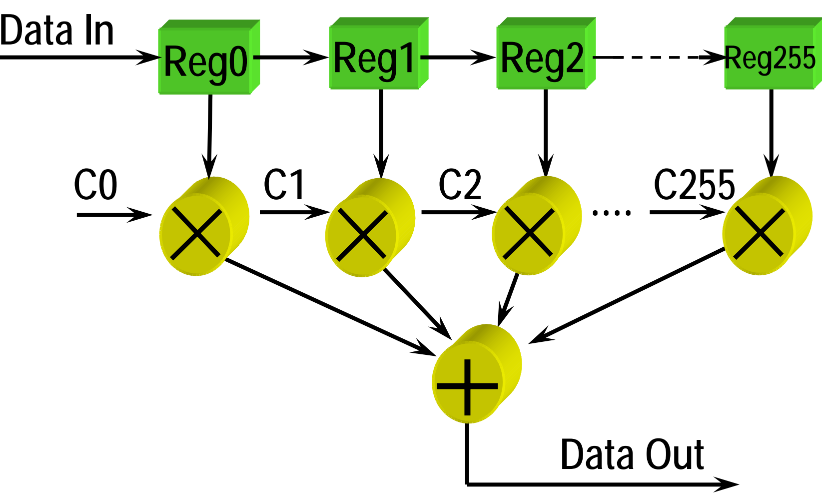
Reason 1: FPGAs handle high computational workloads

Conventional DSP Device
(Von Neumann architecture)



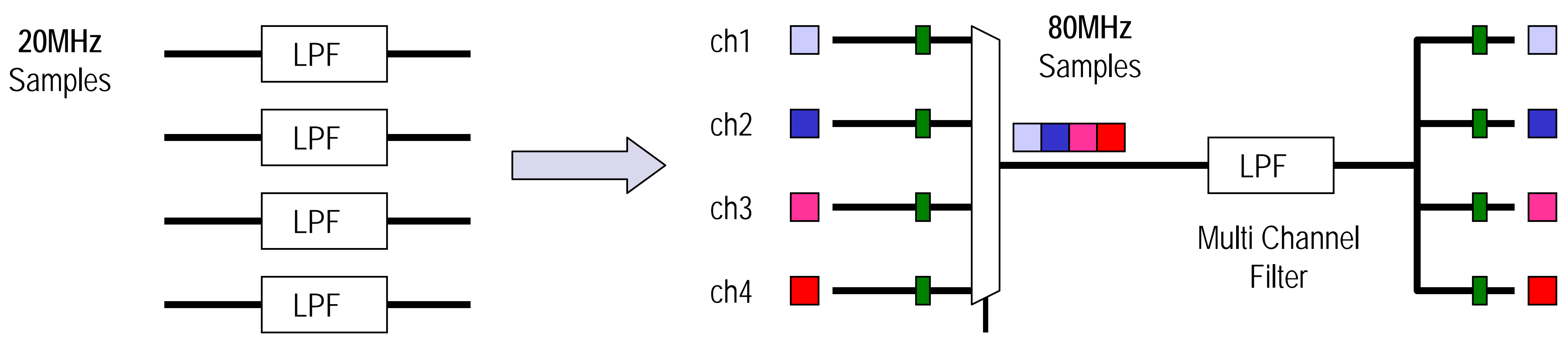
256 Loops needed to process samples

FPGA



All 256 MAC operations in 1 clock cycle

FPGAs are ideal for multi-channel DSP designs



- Many low sample rate channels can be multiplexed (e.g. TDM) and processed in the FPGA, at a high rate.
- Interpolation (using zeros) can also drive sample rates higher.

Why FPGAs for DSP? (2)

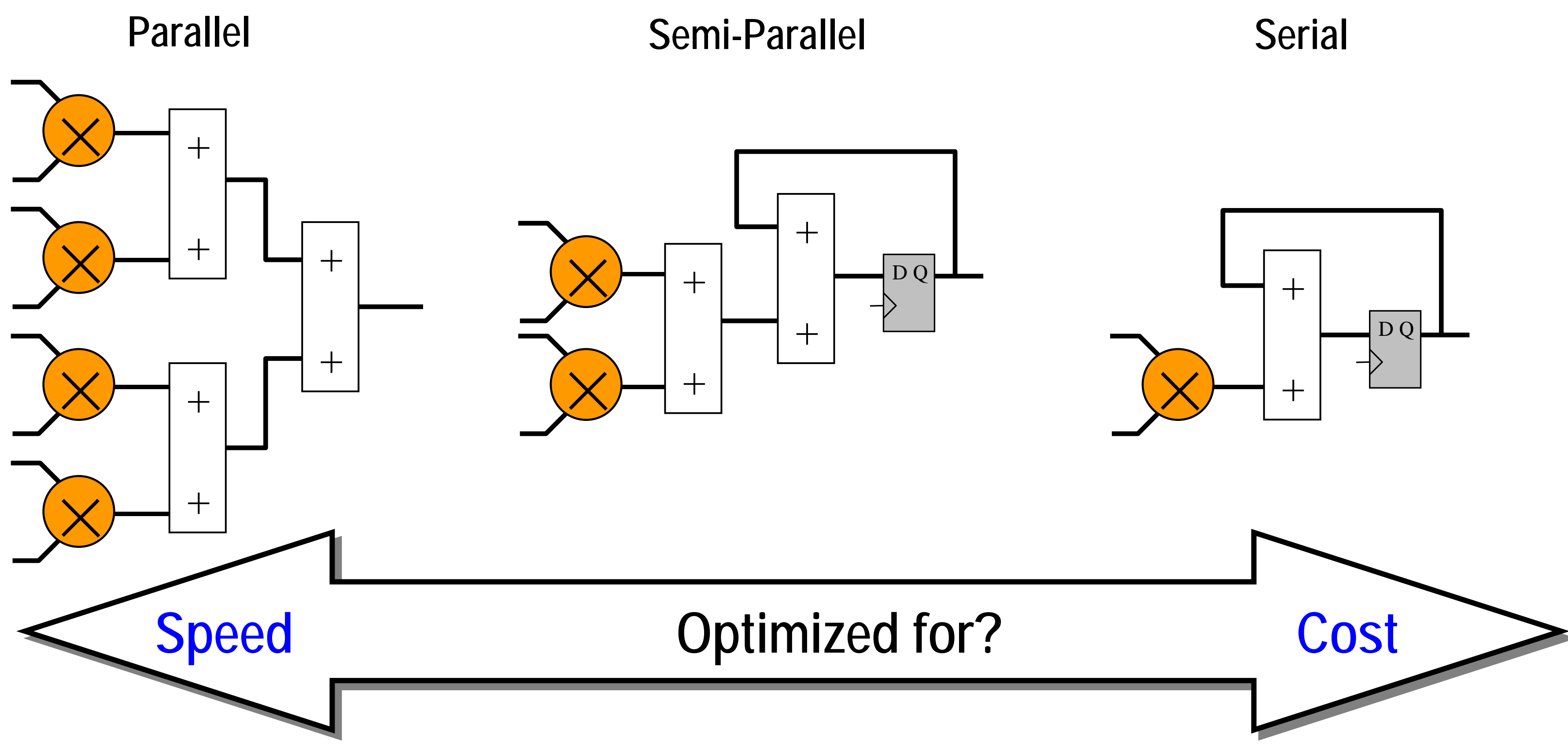
Reason 2: Tremendous Flexibility

Q = (A x B) + (C x D) + (E x F) + (G x H)

can be implemented in parallel

But is this the only way in the FPGA?

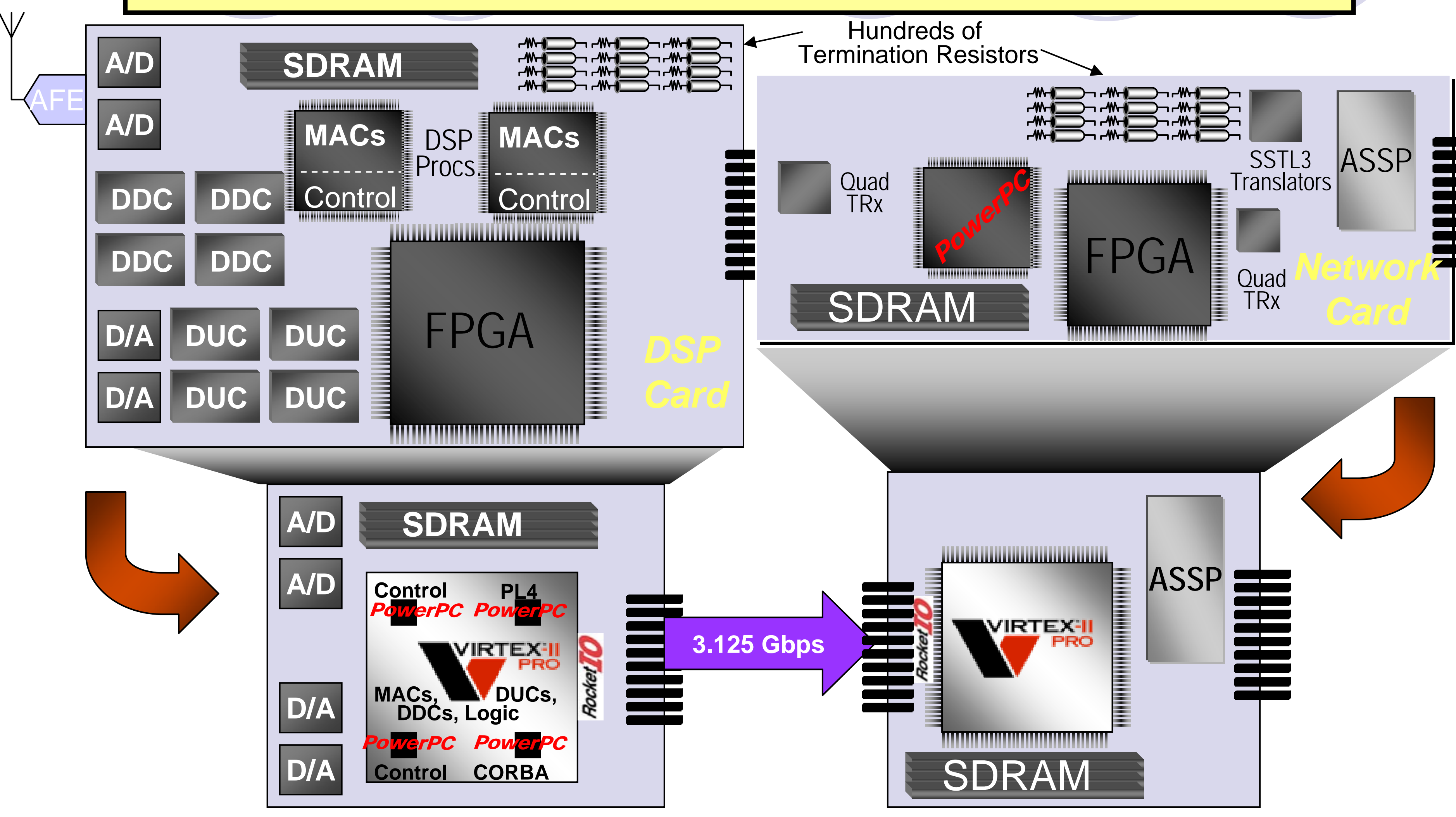
Customize Architectures to Suit Your Ideal Algorithms



FPGAs allow Area (cost) / Performance tradeoffs

Why FPGAs for DSP? (3)

Reason 3: Integration simplifies PCBs



Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

Unsigned integers: positive values only

- ***Unsigned*** integers can be used to represent non-negative numbers. For example using 8 bits we can represent from 0 to 255:

Integer Value	Binary Representation
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
⋮	⋮
64	10000000
65	10000001
⋮	⋮
131	10000011
⋮	⋮
255	11111111

2’s complement

- A more sensible number system for +ve a -ve numbers is 2’s complement which has only one representation of 0 (zero):

Positive Numbers	
Integer	Binary
0	00000000
1	00000001
2	00000010
3	00000011
⋮	⋮
125	01111101
126	01111110
127	01111111

Negative Numbers	
Integer	Binary
0	100000000
-1	11111111
-2	11111110
-3	11111101
⋮	⋮
-125	10000011
-126	10000010
-127	10000001
-128	10000000

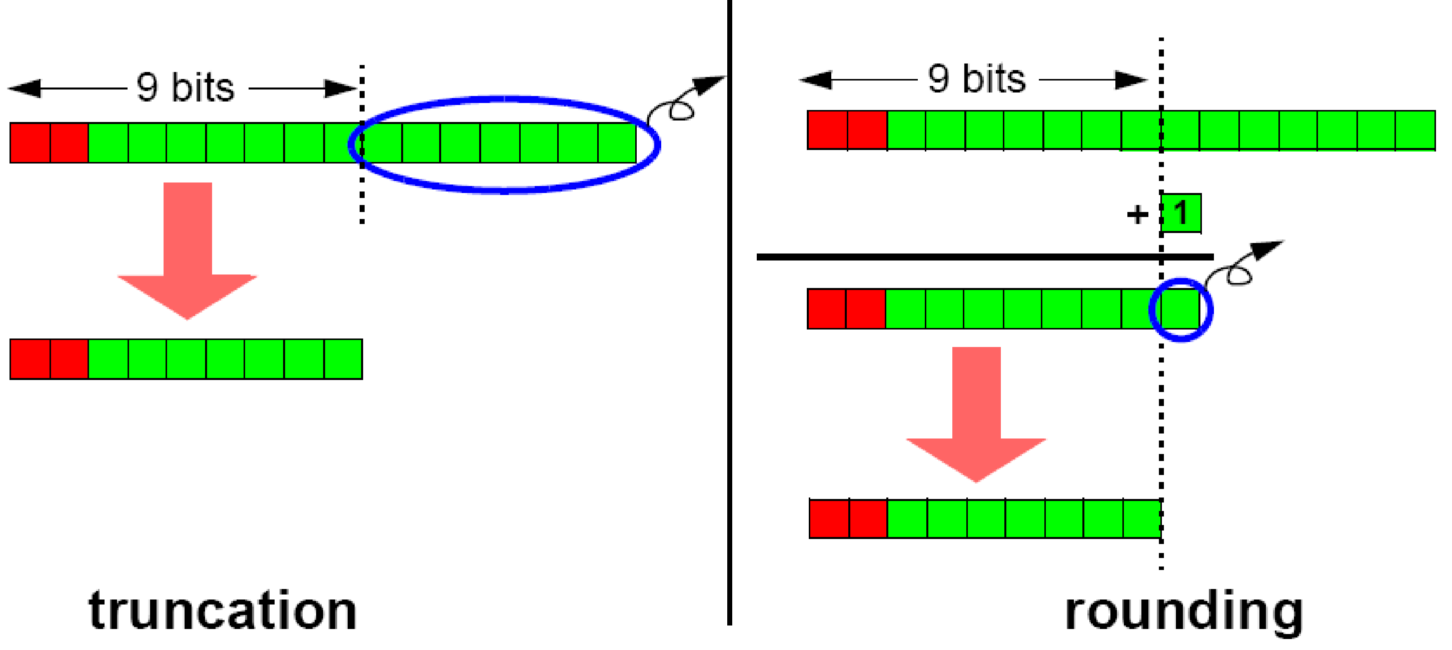
- The 9th bit generated for 0 can be ignored. Note that -128 can be represented but +128 cannot.

Fixed point binary numbers

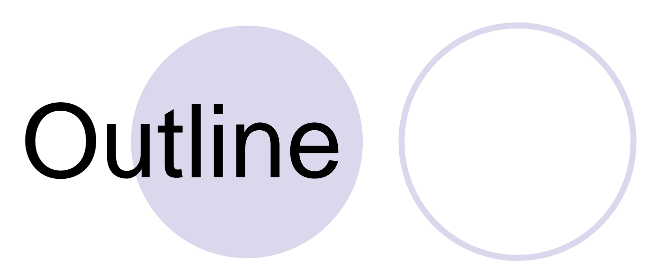
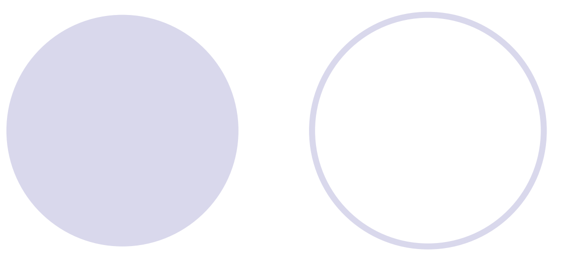
digit worth								decimal value
$-(2^2)$	2^1	2^0 ●	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	
-4	**2**	**1** ●	**0.5**	**0.25**	**0.125**	**0.0625**	**0.03125**	
0	0	0 ●	0	0	0	0	1	0.03125
0	0	0 ●	0	0	0	1	0	0.0625
1	0	1 ●	0	0	0	0	0	-3.0
1	1	0 ●	0	0	1	1	1	-1.78125
1	1	1 ●	1	1	1	1	1	-0.03125

Example: 3 integer bits and 5 fractional bits

Fixed point truncation vs. rounding



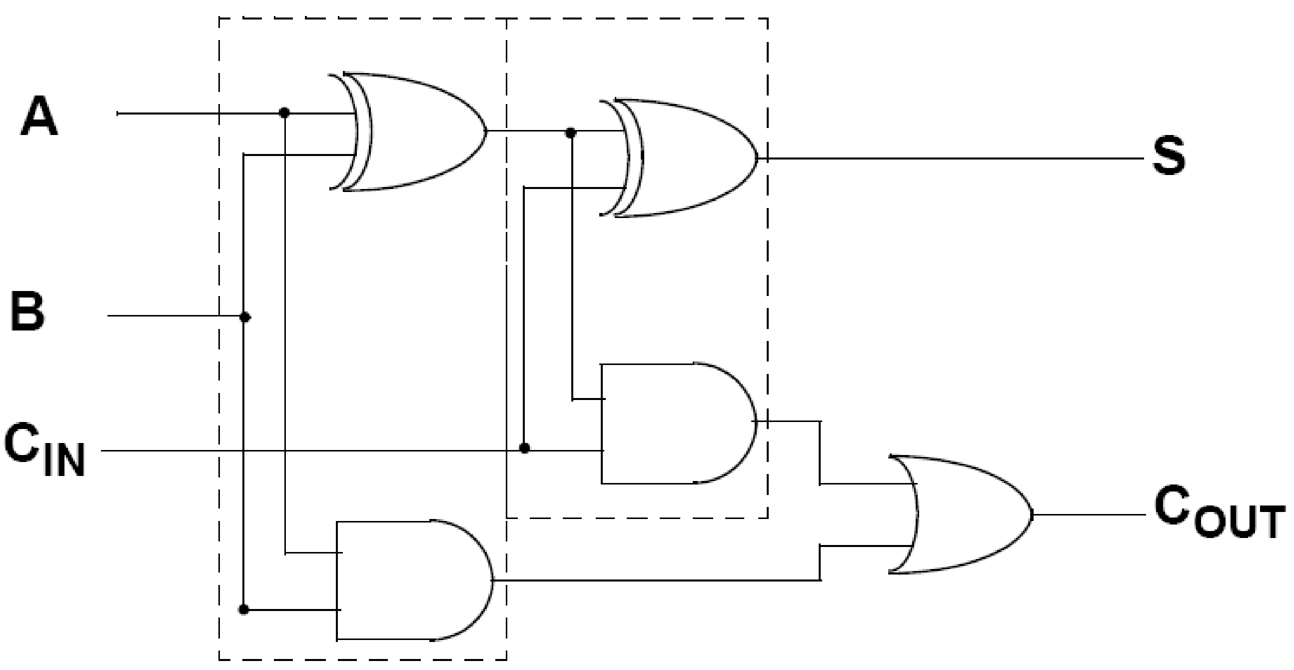
Note that in 2's complement, truncation is biased while rounding isn't.

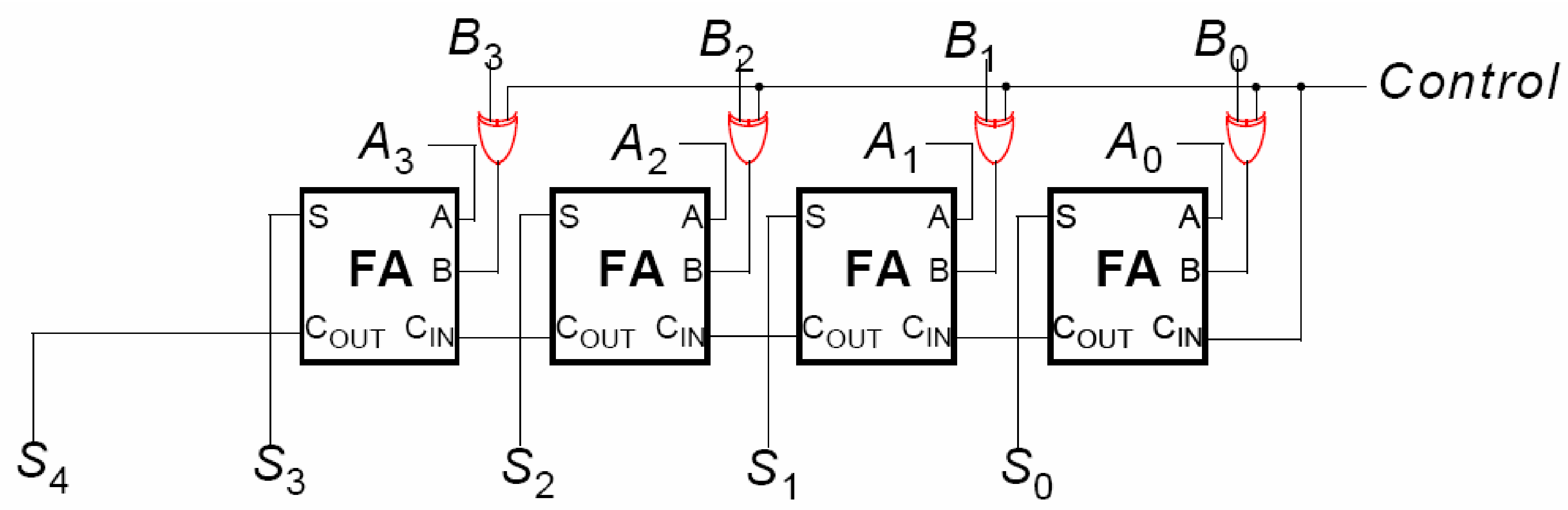
Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

The Full Adder (FA)

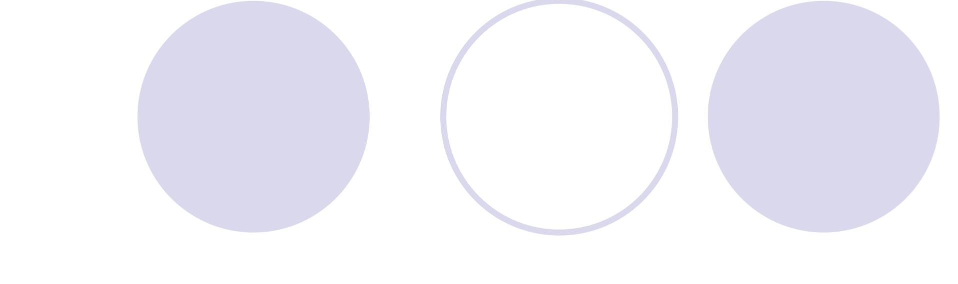


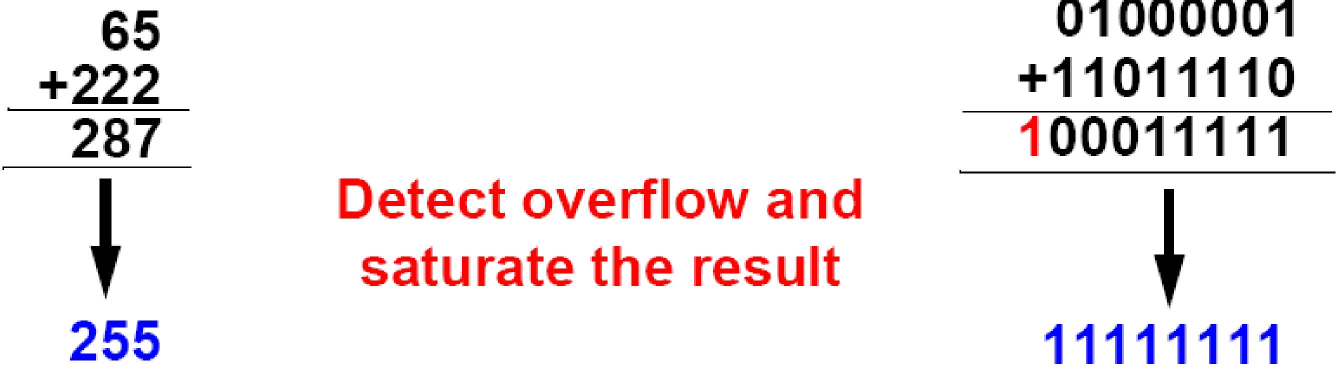
Add/subtract circuit



S = A+B when Control='0'
S = A-B when Control='1'

Saturation



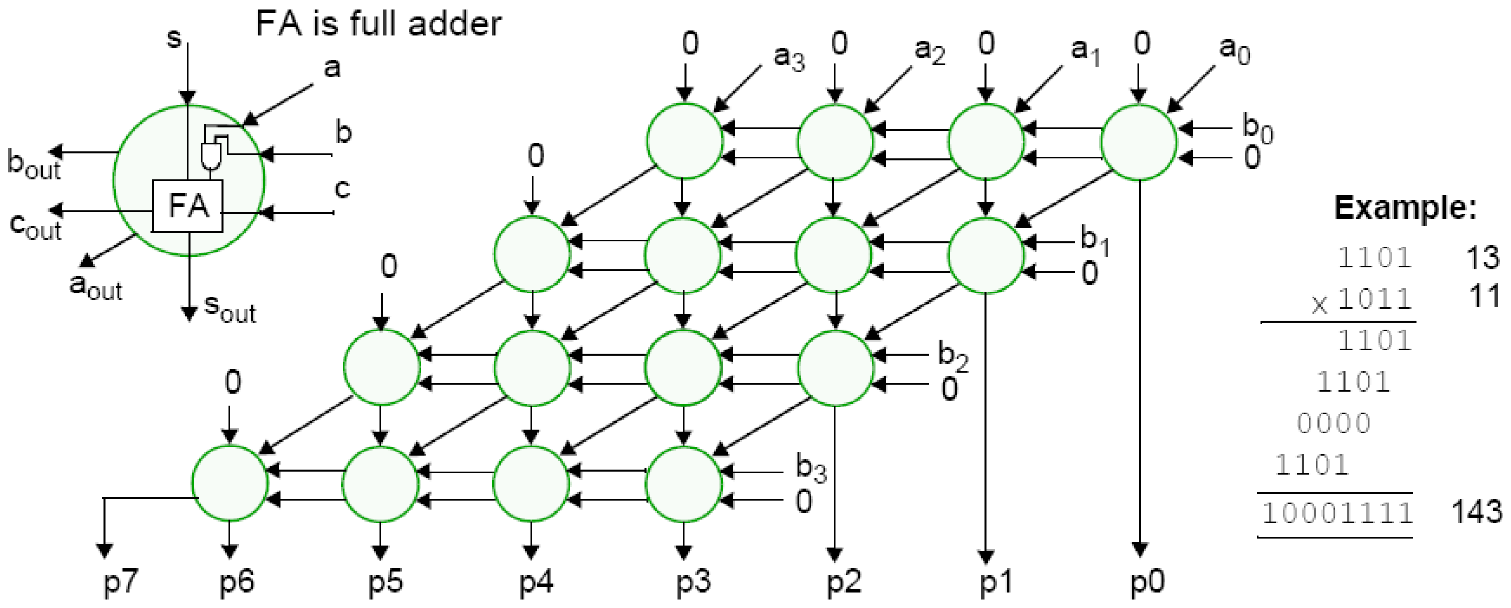


You can't let the data path become arbitrarily wide. Saturation involves overflow detection and a multiplexer. Useful in accumulators (like the one in the PI controller we use in the lab).

Multiplication: pencil & paper approach

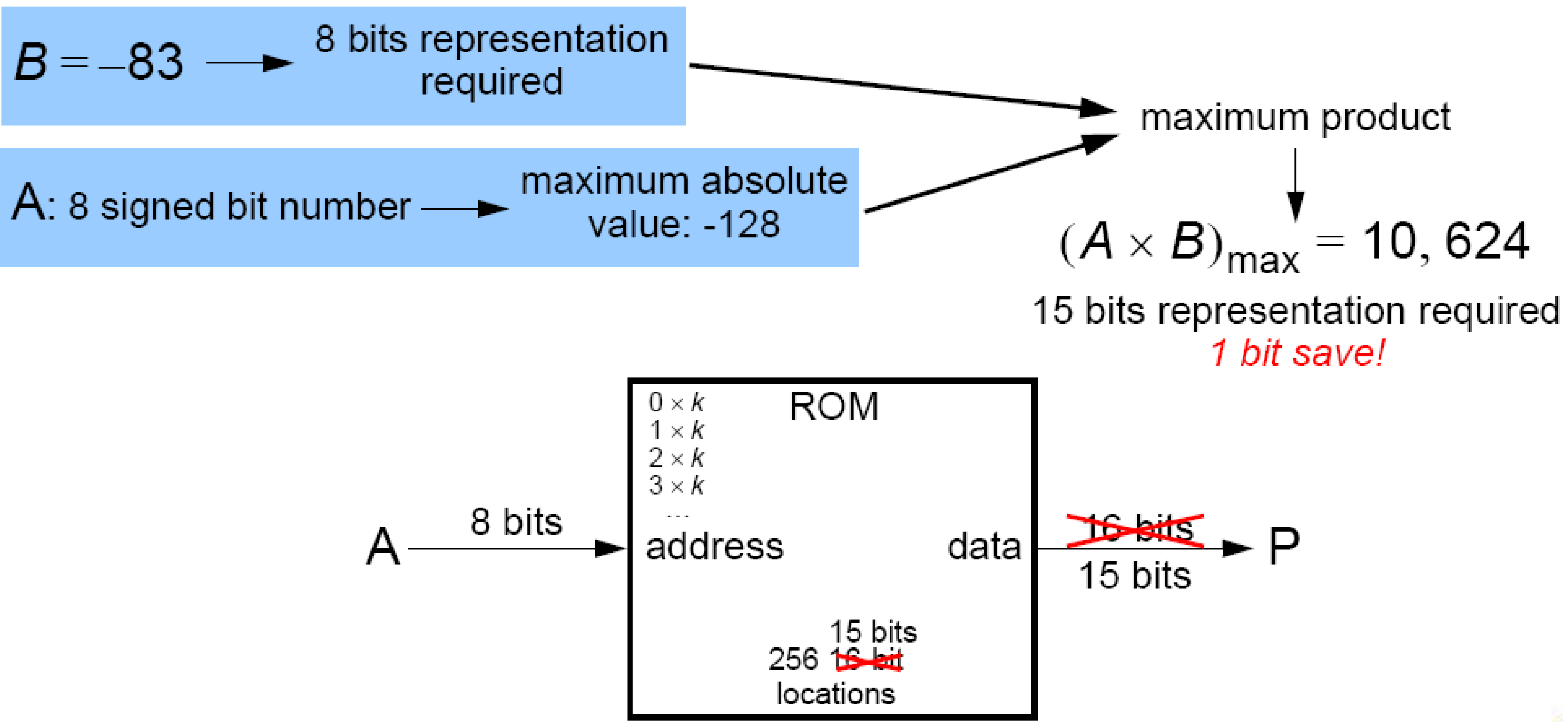
```
         11010110   A7...A0
        x00101101   B7...B0
-------------------
         11010110
        000000000
       1101011000
      11010110000
     000000000000
    1101011000000
   00000000000000
  000000000000000
-------------------
 0010010110011110   P15...P0
```

A 4-bit unsigned multiplier using Full Adders and AND gates



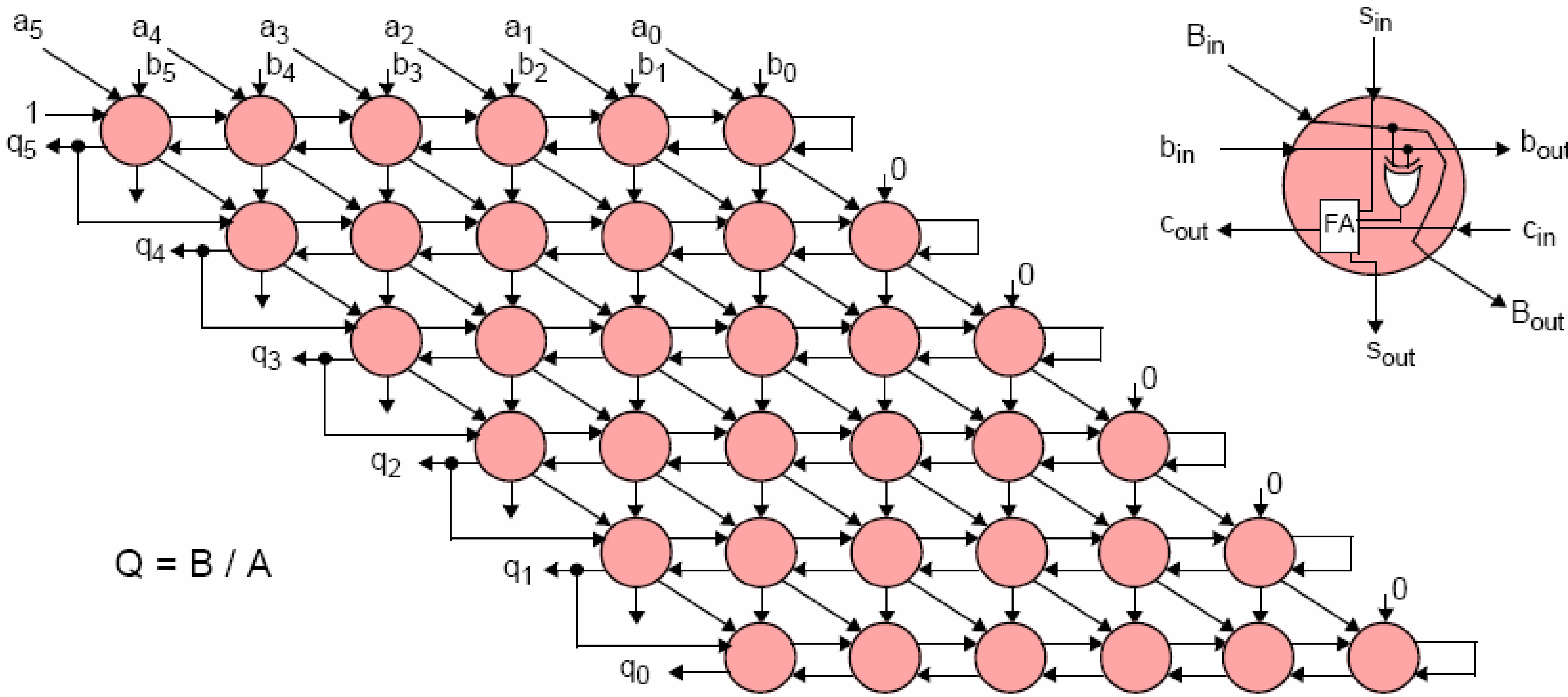
Of course, you can use embedded multipliers if your chip has them!

Constant coefficient multipliers using ROM



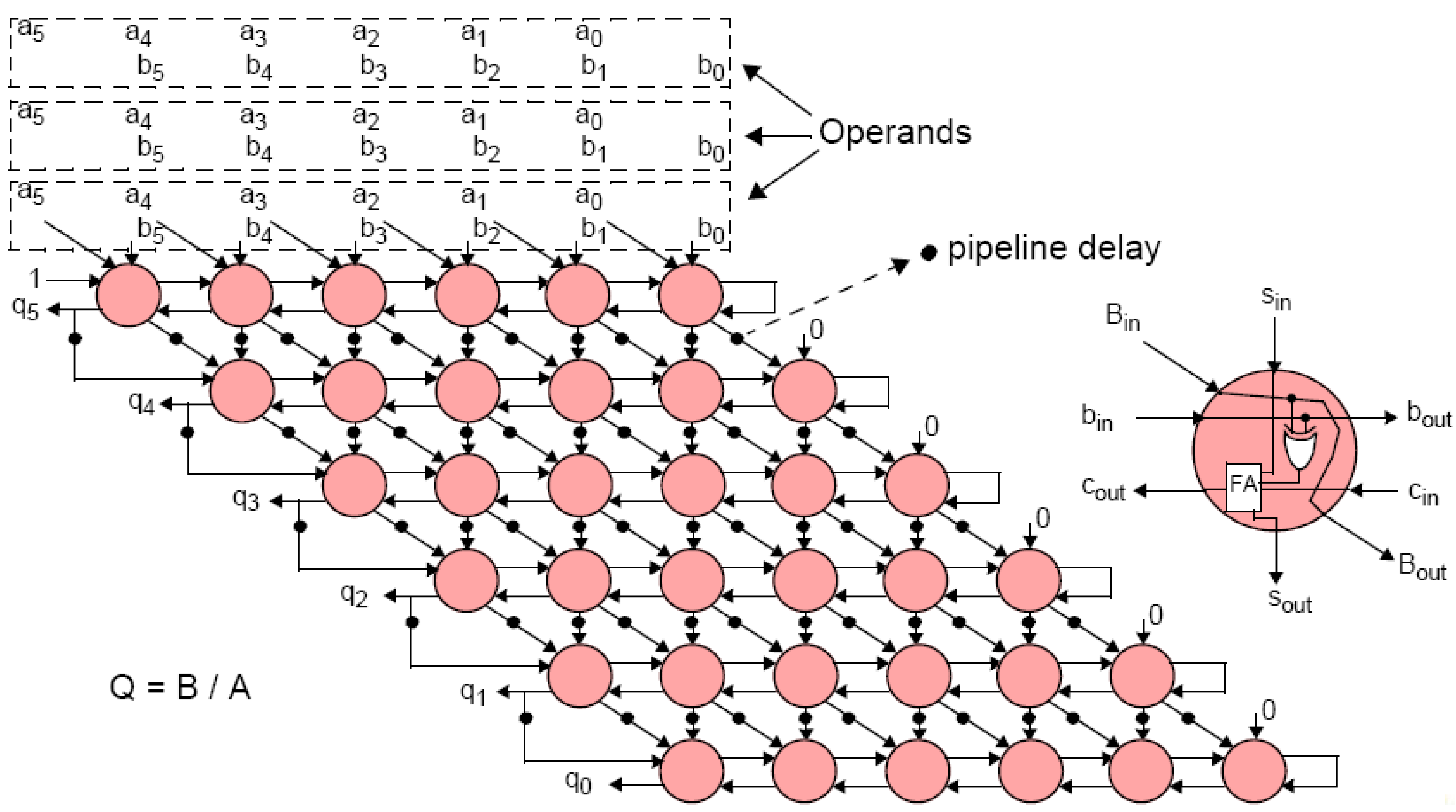
For “easy” coefficients, there are smarter ways. E.g. to multiply a number A by 31, left-shift A by 5 places then subtract A.

Division: pencil & paper

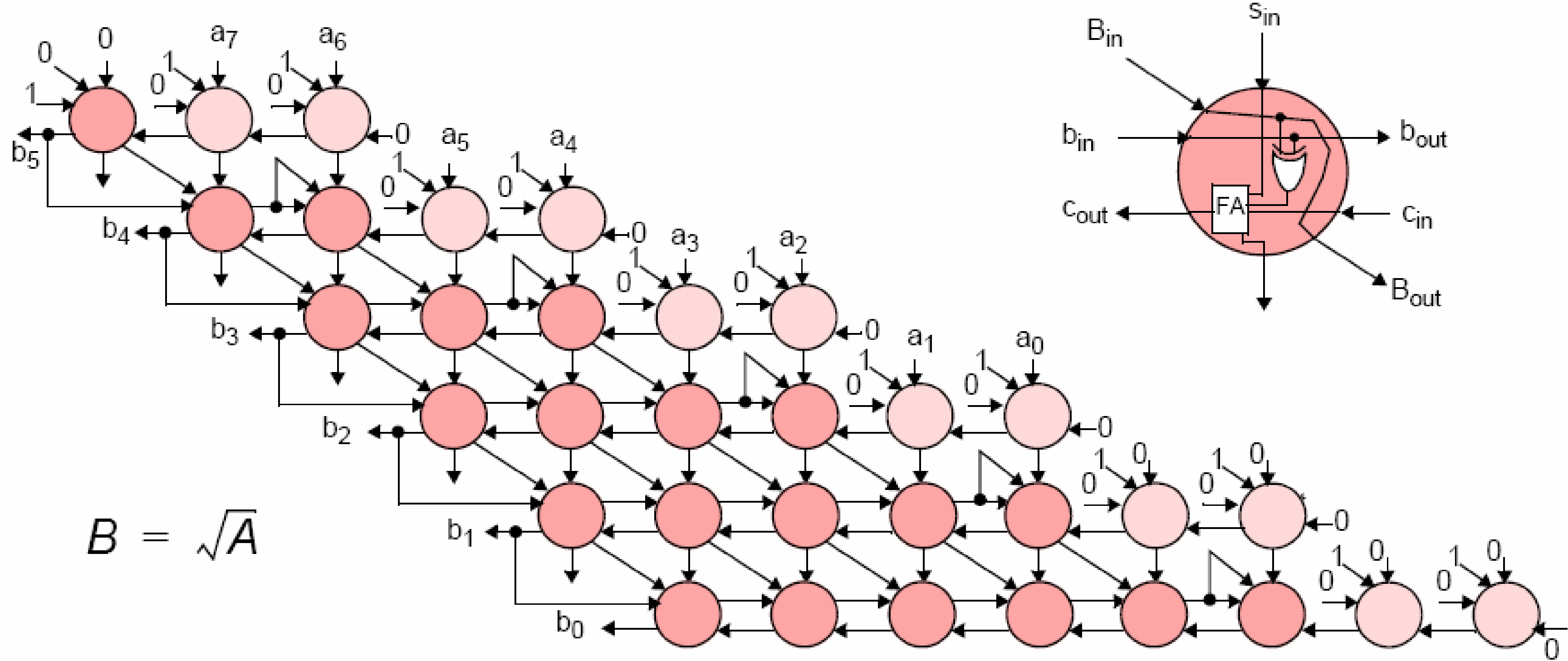


- Uses add/subtract blocks presented earlier.
- MSB produced first: this will usually imply we have to wait for whole operation to finish before feeding result to another block.
- Longer combinational delays than in multiplication: an N by N division will always take longer than an N by N multiplication.

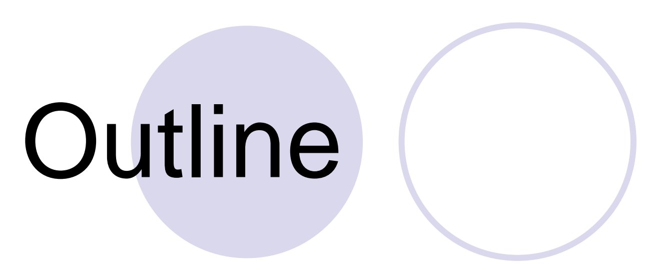
Pipelining the division array

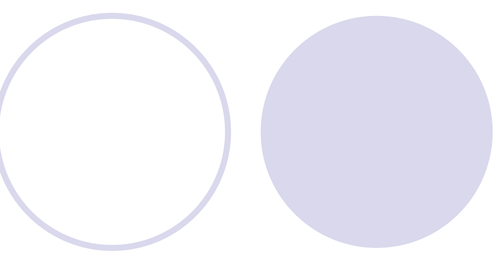


Square root



- Take a division array, cut it in half (diagonally) and you have square root. Square root is therefore faster than division!
- Although with less ripple through, this block suffers from the same problems as the division array.
- Alternative approach: first guess with a ROM, then use an iterative algorithm such as Newton-Raphson.





Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

Distributed Arithmetic (DA) 1/2

Digital filtering is about sums of products:

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

Let's assume:
- c[n] constant (prerequisite to use DA)
- x[n] input signal B bits wide

Then:

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

$x_b[n]$ is bit number b of x[n] (either 0 or 1)

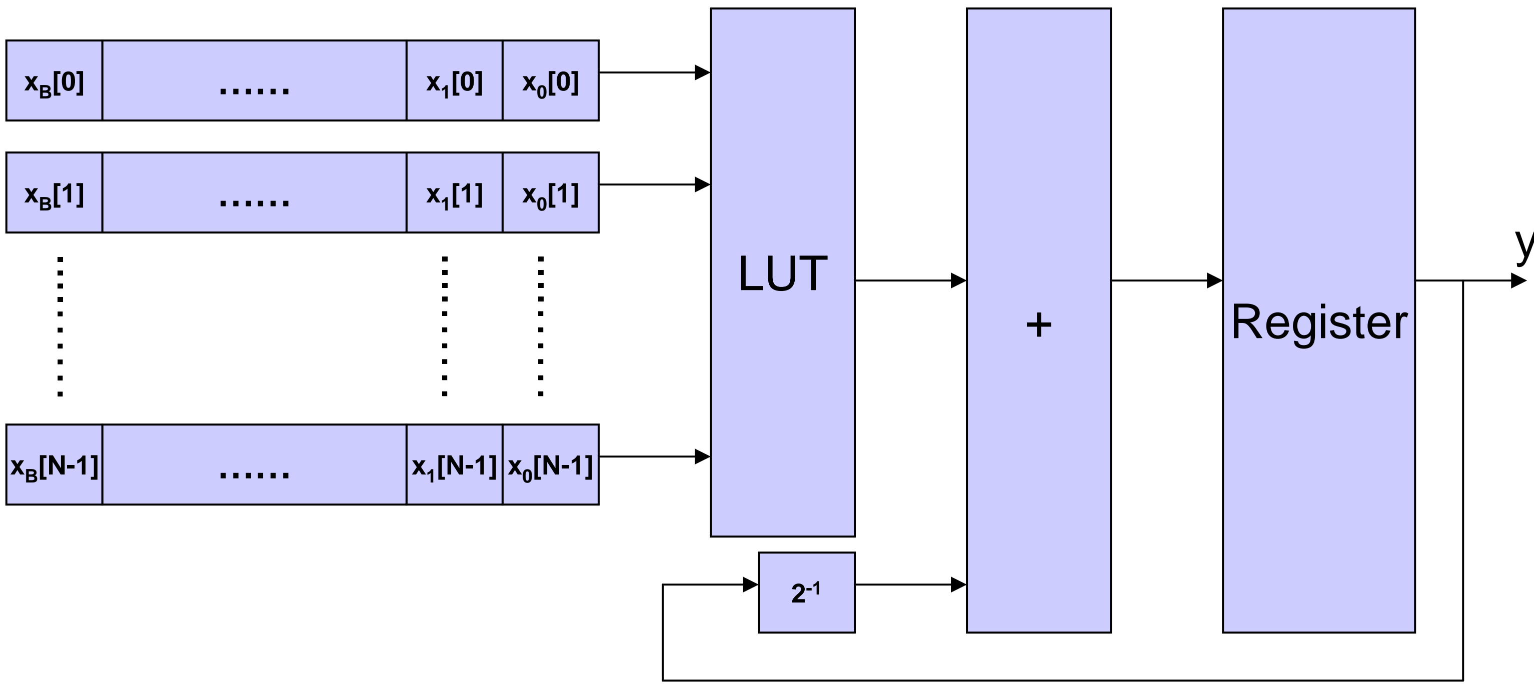
And after some rearrangement of terms:

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

This can be implemented with an N-input LUT

Distributed Arithmetic (DA) 2/2

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



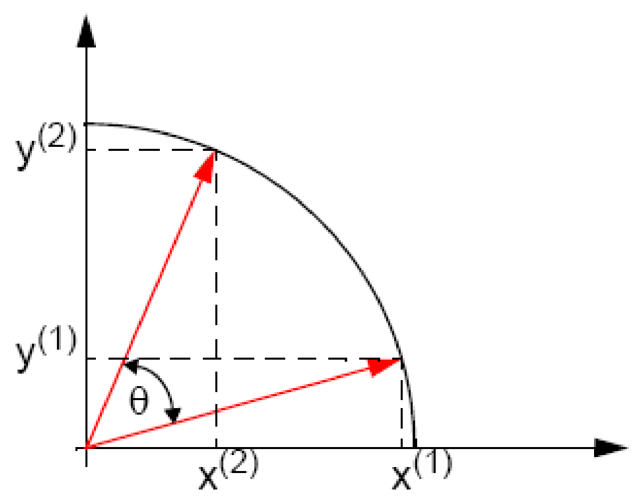
Generates a result every B clock ticks. Replicating logic one can trade off speed vs. area, to the limit of getting one result per clock tick.

Outline

- Digital Signal Processing using FPGAs
  - Introduction. Why FPGAs for DSP?
  - Fixed point and its subtleties.
  - Doing arithmetic in hardware.
  - Distributed Arithmetic (DA).
  - COordinate Rotation DIgital Computer (CORDIC).

COrdinate Rotation DIgital Computer

- The CORDIC method is based on the rotation of a vector from position $(x^{(1)}, y^{(1)})$ to $(x^{(2)}, y^{(2)})$:



- The new position can be calculated using the Givens rotation:

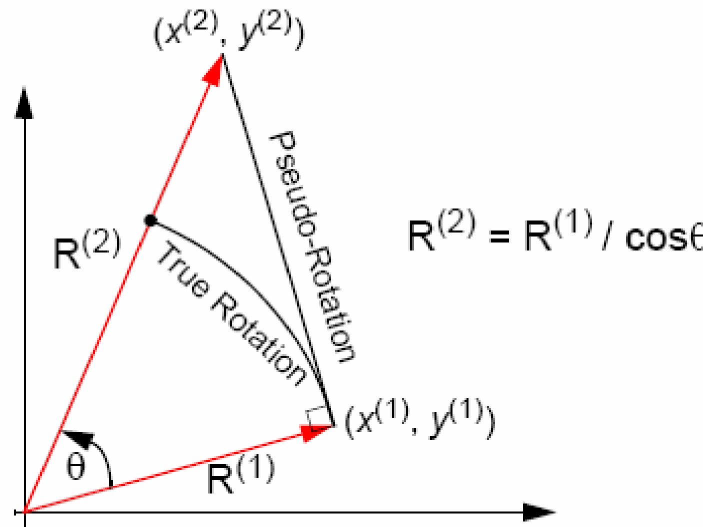
$$x^{(2)} = x^{(1)}\cos\theta - y^{(1)}\sin\theta = \cos\theta(x^{(1)} - y^{(1)}\tan\theta)$$

$$y^{(2)} = x^{(1)}\sin\theta + y^{(1)}\cos\theta = \cos\theta(y^{(1)} + x^{(1)}\tan\theta)$$

Pseudo-rotations

- By removing the $\cos\theta$ term, the equations give the result of a Pseudo-Rotation:

$$x^{(2)} = x^{(1)} - y^{(1)}\tan\theta$$
$$y^{(2)} = y^{(1)} + x^{(1)}\tan\theta$$



Basic CORDIC iterations

- The key to the CORDIC method is to only rotate by angles of θ where $\tan\theta^i = 2^{-i}$ $\Rightarrow$ multiplication by tangent term becomes a shift!

- The table below shows the first few rotation angles that must be used for each iteration (i) of the CORDIC algorithm:

i	θ^i	$\tan\theta^i = 2^{-i}$
0	45	1
1	26.6	0.5
2	14	0.25
3	7.1	0.125
4	3.6	0.0625

- Thus rotating by an arbitrary angle θ now becomes an iterative process made up of successively smaller pseudo-rotations.

Angle accumulator

- The simplified Givens transform shown earlier can now be expressed for each iteration as:

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$
$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

- At this stage we introduce a 3rd equation called the Angle Accumulator which is used to keep track of the accumulative angle rotated at each iteration:

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)} \qquad \text{(Angle Accumulator)}$$

where d_i = +/- 1

- The symbol d_i is a decision operator and is used to decide which direction to rotate.

The scaling factor

- The Scaling Factor is a by-product of the pseudo-rotations.
- When simplifying the algorithm to allow pseudo-rotations the $\cos\theta$ term was omitted.
- Thus outputs $x^{(n)}$, $y^{(n)}$ are scaled by a factor K_n where:

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n (\sqrt{1 + 2^{(-2i)}})$$

- However if the number of iterations are known then the Scaling Factor K_n can be precomputed.
- Also, $1/K_n$ can be precomputed and used to calculate the true values of $x^{(n)}$ and $y^{(n)}$.

Rotation Mode

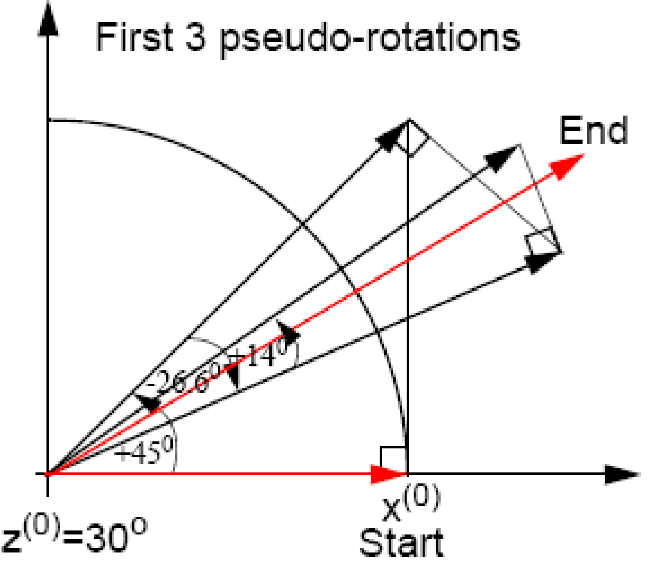
- The CORDIC method is operated in one of two modes;
- The mode of operation dictates the condition for the control operator d_i;
- In Rotation Mode choose: $d_i = \text{sign}(z^{(i)}) \Rightarrow z^{(i)} \rightarrow 0$;
- After n iterations we have:

$$x^{(n)} = K_n(x^{(0)} \cos z^{(0)} - y^{(0)} \sin z^{(0)})$$
$$y^{(n)} = K_n(y^{(0)} \cos z^{(0)} + x^{(0)} \sin z^{(0)})$$
$$z^{(n)} = 0$$

- Can compute $\cos z^{(0)}$ and $\sin z^{(0)}$ by starting with $x^{(0)} = 1/K_n$ and $y^{(0)} = 0$

Example: calculate sin and cos of 30°

i	d_i	$\theta^{(i)}$	$z^{(i)}$	$y^{(i)}$	$x^{(i)}$
0	+1	45	+30	0	0.6073
1	-1	26.6	-15	0.6073	0.6073
2	+1	14	+11.6	0.3036	0.9109
3	-1	7.1	-2.4	0.5313	0.8350
4	+1	3.6	+4.7	0.4270	0.9014
5	+1	1.8	+1.1	0.4833	0.8747
6	-1	0.9	-0.7	0.5106	0.8596
7	+1	0.4	+0.2	0.4972	0.8676
8	-1	0.2	-0.2	0.5040	0.8637
9	+1	0.1	+0	0.5006	0.8657



Vectoring Mode

- In Vectoring Mode choose: $d_i = -\text{sign}(x^{(i)}y^{(i)}) \Rightarrow y^{(i)} \rightarrow 0$
- After n iterations we have:

$$x^{(n)} = K_n\left(\sqrt{(x^{(0)})^2 + (y^{(0)})^2}\right)$$

$$y^{(n)} = 0$$

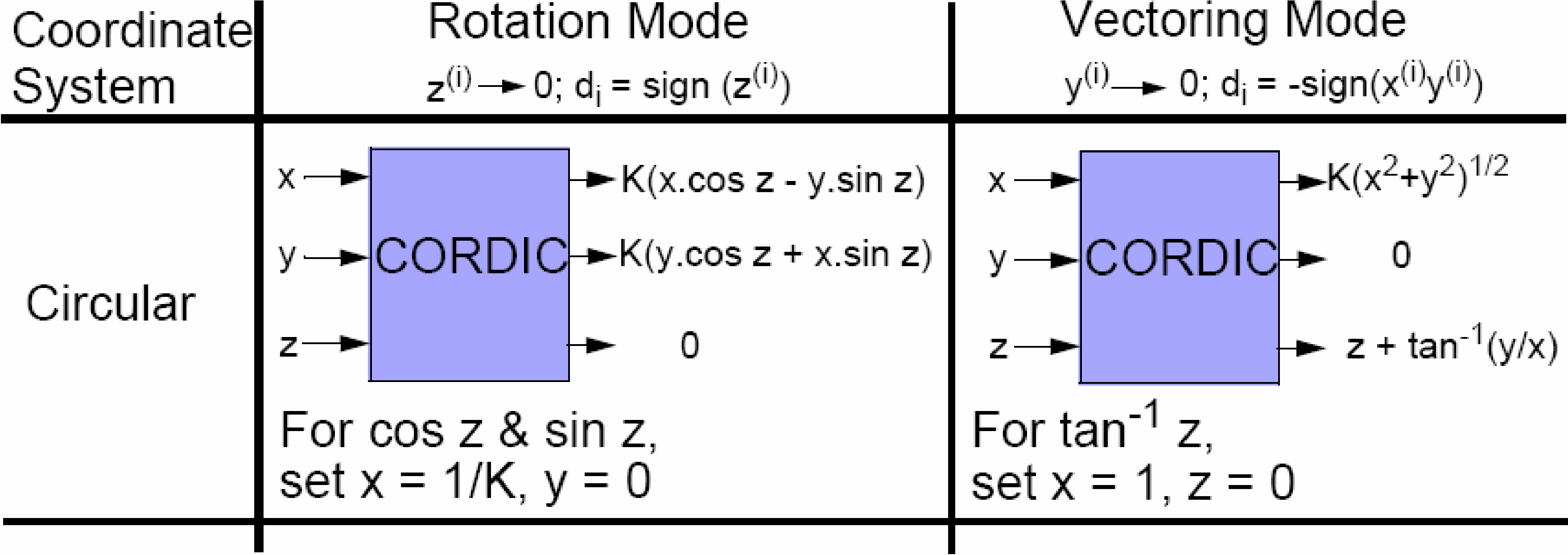
$$z^{(n)} = z^{(0)} + \tan^{-1}\left(\frac{y^{(0)}}{x^{(0)}}\right)$$

Vector magnitude

- Can compute $\tan^{-1} y^{(0)}$ by setting $x^{(0)} = 1$ and $z^{(0)} = 0$

Circular coordinate system

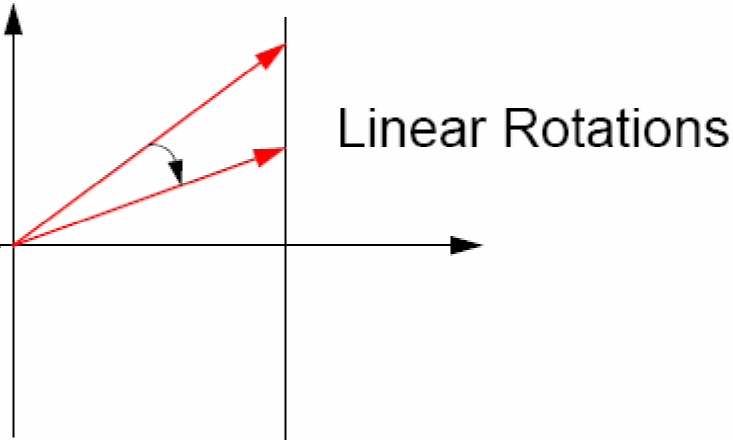
- So far only pseudo-rotations in a Circular Coordinate System have been considered.
- Thus, the following functions can be computed:



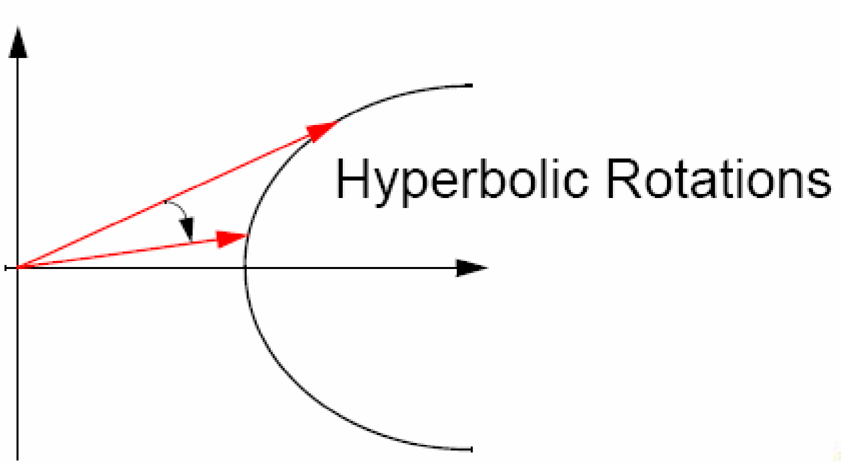
- However, more functions can be computed if we use other coordinate systems.

Other coordinate systems

- Linear Coordinate System



- Hyperbolic Coordinate System



Generalized CORDIC equations

- With the addition of two other Coordinate Systems the CORDIC equations can now be generalised to accommodate all three systems:

$$x^{(i+1)} = (x^{(i)} - \mu d_i (2^{-i} y^{(i)}))$$

$$y^{(i+1)} = (y^{(i)} + d_i (2^{-i} x^{(i)}))$$

$$z^{(i+1)} = z^{(i)} - d_i e^{(i)}$$

- Circular Rotations: $\mu = 1, e^{(i)} = \tan^{-1} 2^{-i}$
- Linear Rotations: $\mu = 0, e^{(i)} = 2^{-i}$
- Hyperbolic Rotations: $\mu = -1, e^{(i)} = \tanh^{-1} 2^{-i}$

Summary of CORDIC functions

	Rotation Mode: d_i=sign($z^{(i)}$); $z^{(i)} \rightarrow 0$	Vectoring Mode: d_i=-sign($x^{(i)}y^{(i)}$); $y^{(i)} \rightarrow 0$
Circular $\mu = 1$ $e^{(i)} = \tan^{-1}2^{-i}$	x → CORDIC → K(x.cos z - y.sin z) y → CORDIC → K(y.cos z + x.sin z) z → CORDIC → 0 For cos z & sin z, set x = 1/K, y = 0	x → CORDIC → $K(x^2 + y^2)^{1/2}$ y → CORDIC → 0 z → CORDIC → $z + \tan^{-1}(y/x)$ For $\tan^{-1}$ y, set x = 1, z = 0
Linear $\mu = 0$ $e^{(i)} = 2^{-i}$	x → CORDIC → x y → CORDIC → y + (x.z) z → CORDIC → 0 For multiplication, set y = 0	x → CORDIC → x y → CORDIC → 0 z → CORDIC → z + (y/x) For division, set z = 0
Hyperbolic $\mu = -1$ $e^{(i)} = \tanh^{-1}2^{-i}$	x → CORDIC → K^*(x.cosh z - y.sinh z) y → CORDIC → K^*(y.cosh z + x.sinh z) z → CORDIC → 0 For cosh z & sinh z, set x = 1/K^*, y = 0	x → CORDIC → $K^*(x^2 - y^2)^{1/2}$ y → CORDIC → 0 z → CORDIC → $z + \tanh^{-1}(y/x)$ For $\tanh^{-1}$y, set x = 1, z = 0

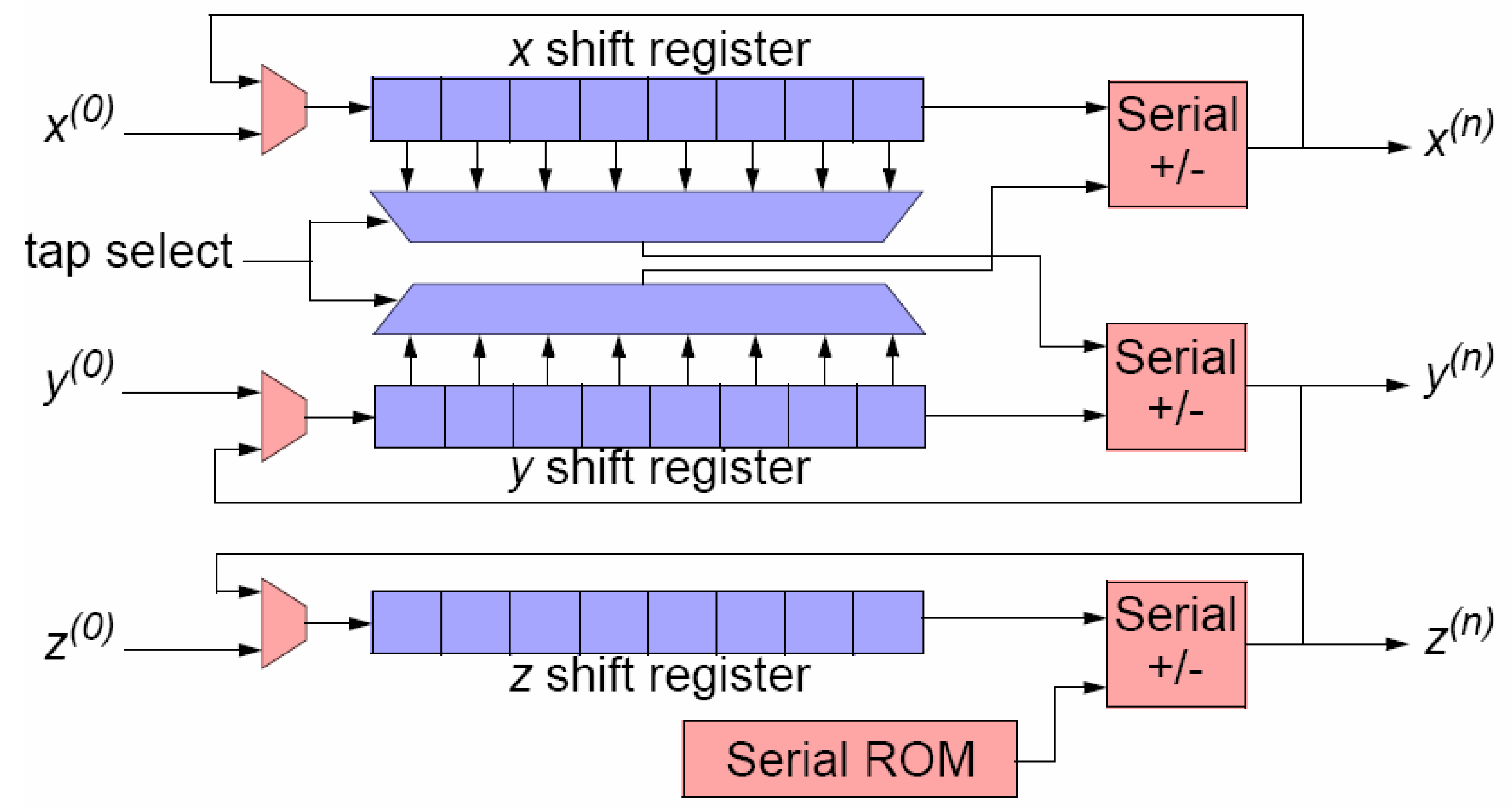
Precision and convergence

- For k bits of precision in trigonometric functions, k iterations are required.
- Convergence is guaranteed for Circular & Linear CORDIC using angles in range $-99.7 \leq z \leq 99.7$:
    - for angles outside this range use standard trig identities.
- Elemental rotations using Hyperbolic CORDIC do not converge:
    - convergence is achieved if certain iterations are repeated;
    - $i = 4, 13, 40, \ldots\ldots, k, 3k+1, \ldots$

FPGA implementation

- The ideal CORDIC architecture depends on speed vs area tradeoffs in the intended application.
- A direct translation of the CORDIC equations is an iterative bit-parallel design, however:
    - bit-parallel variable shift shifters do not map well into FPGAs;
    - require several FPGA cells resulting in large, slow design.
- We shall consider an iterative bit-serial solution to illustrate:
    - a minimum area architecture;
    - one implementation of variable shift shifters.

Iterative bit-serial design



Acknowledgements

- Many thanks to Jeff Weintraub (Xilinx University Program) and Bob Stewart (University of Strathclyde) for many of these slides.