6.S965
Digital Systems Laboratory II

Lecture 7:

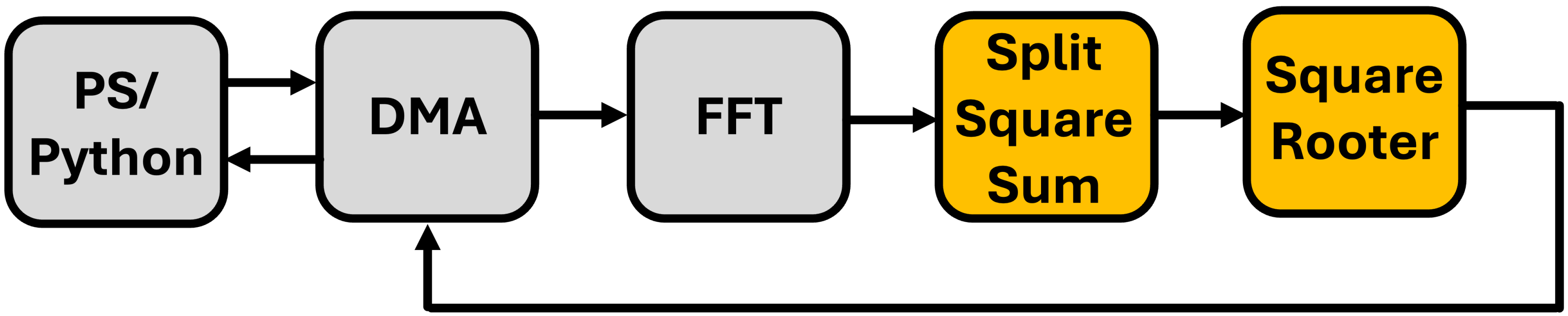
CORDIC and Iterative Algorithms

Administrative

- Week 4 material is on site. Sorry for delay. Due a week from today.
- Week 5 will come out on Friday.

Week 4 Stuff

- Build two modules and integrate together into another DMA pipeline



Where we are/where we're going…

- Last week you all got to mess with the DMA and build a FIR and just get some reps in with the Pynq framework.
- We'll do one or two more labs with the Pynq Z2 board then move to the RFSoC for a couple weeks (I think)…then be into final projects

For Final Projects

- I'd strongly encourage you to try to build as much as possible from your designs
- Don't necessarily go for the high-level cool stuff…making a full pipeline from scratch (no IP or anything…) can be really cool

There are tons of cool algorithms out there

- Particularly for FPGAs or digital environments in general

Low-cost, High-speed Parallel FIR Filters for RFSoC Front-Ends Enabled by CλaSH

Craig Ramsay
University of Strathclyde
Glasgow, Scotland
craig.ramsay.100@strath.ac.uk

Louise H. Crockett
University of Strathclyde
Glasgow, Scotland
louise.crockett@strath.ac.uk

Robert W. Stewart
University of Strathclyde
Glasgow, Scotland
r.stewart@strath.ac.uk

***Abstract*—We present a new low-cost, high-speed parallel FIR filter generator targeting the Xilinx Radio Frequency System on Chip (RFSoC) and direct RF sampling applications. We compose two existing approaches in a novel hierarchy: efficient parallelism with Fast FIR Algorithm (FFA) structures, and efficient multiplierless FIR implementations with H_{cub}. The resource usage advantages (in both area and type) are compared with similar output from the traditional architecture, exemplified by vendor tools, as well as the H_{cub}-based filters without the FFA optimisation. Although these techniques are well studied individually in the literature, they have not enjoyed mainstream use as their structural complexity proves awkward to capture with traditional Hardware Description Languages (HDLs). This work continues a discussion of the use of functional programming techniques in hardware description, highlighting the benefits of having easily composable circuit generators.**

I. Introduction

We present a new family of low-cost, high-speed, parallel Finite Impulse Response (FIR) filters targeting direct Radio Frequency (RF) sampling applications with the Xilinx Zynq

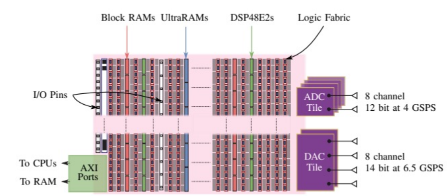


Fig. 1. Overview of RFSoC's FPGA and RF Data Converters

- Custom Digital Up/Down Conversion (DUC/DDC) as a front-end of *any* radio application. Especially useful when the characteristics of the available hardened DUC/DDCs [5] do not meet the application's requirements.

The demand for sample parallelism and the multi-channel nature of the RFSoC device amplifies the effects of filter

CORDIC

- **Co**ordinate **R**otation **Di**gital **C**omputer

- Super versatile class of iterative algorithms that are used widely in hardware because they are relatively simple to implement

- Might not be the fastest, but are a good gateway algorithm for lots of options out there.

CORDIC

- What can you compute with CORDIC?

Directly computable functions [edit | edit source]

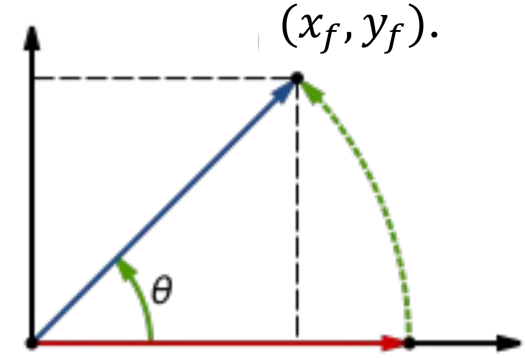
$\sin z$	$\cos z$
$\tan^{-1} z$	$\sinh z$
$\cosh z$	$\tanh^{-1} z$
y/x	xz
$\tan^{-1}(y/x)$	$\sqrt{x^2+y^2}$
$\sqrt{x^2-y^2}$	$e^z = \sinh z + \cosh z$

Indirectly computable functions [edit | edit source]

In addition to the above functions, a number of other functions can be produced by combining the results of previous computations:

$\tan z = \frac{\sin z}{\cos z}$	$\cos^{-1} w = \tan^{-1} \frac{\sqrt{1-w^2}}{w}$
$\tanh z = \frac{\sinh z}{\cosh z}$	$\sin^{-1} w = \tan^{-1} \frac{w}{\sqrt{1-w^2}}$
$\ln w = 2\tanh^{-1} \frac{w-1}{w+1}$	$\log_b w = \frac{\ln w}{\ln b}$
$w^t = e^{t \ln w}$	$\cosh^{-1} = \ln\left(w + \sqrt{w^2-1}\right)$
$\tan^{-1}(y/x)$	$\sinh^{-1} = \ln\left(w + \sqrt{w^2+1}\right)$
$\sqrt{x^2-y^2}$	$\sqrt{w} = \sqrt{(w+1/4)^2-(w-1/4)^2}$

CORDIC



- Built around the idea of rotations

- Rotation Matrix:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- Also break down into two equations:

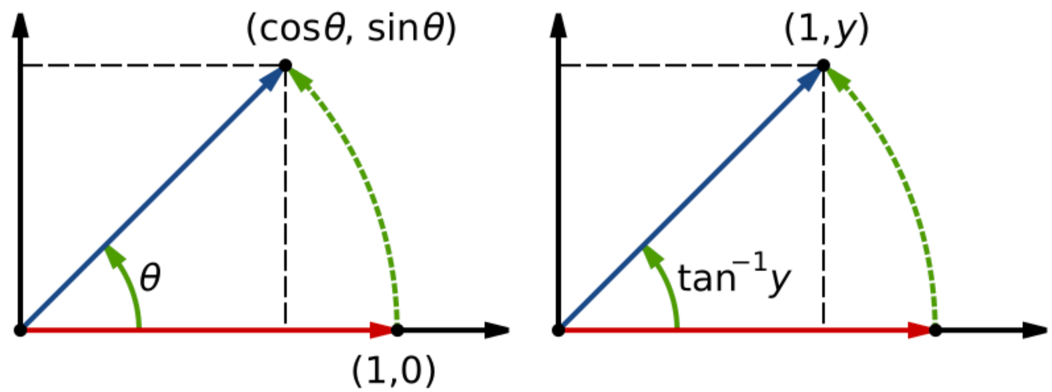
$$x_f = \cos(\theta)\, x_i - \sin(\theta)\, y_i$$

$$y_f = \sin(\theta)\, x_i + \cos(\theta)\, y_i$$

https://zipcpu.com/dsp/2017/08/30/cordic.html https://en.wikibooks.org/wiki/Digital_Circuits/CORDIC

As Motivation to do this…

- If we could carry out that rotation we could start to answer questions like…



- Start at (1, 0)
- Rotate by θ
- We get $(\cos\theta, \sin\theta)$

- Start at $(1, y)$
- Rotate until $y = 0$
- The rotation is $\tan^{-1} y$

OK so what do we need to do…

- We need to be able to do this…

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- But this is a little chicken-and-egg…because in order to do this, we need to be able to do $\sin(\theta)$ or $\cos(\theta)$ which are things we don't have as ready-made functions

Trig Identities

Trigonometric functions and their reciprocals on the unit circle. All of the right-angled triangles are similar, i.e. the ratios between their corresponding sides are the same. For sin, cos and tan the unit-length radius forms the hypotenuse of the triangle that defines them. The reciprocal identities arise as ratios of sides in the triangles where this unit line is no longer the hypotenuse. The triangle shaded blue illustrates the identity $1 + \cot^2\theta = \csc^2\theta$, and the red triangle shows that $\tan^2\theta + 1 = \sec^2\theta$.

Each trigonometric function in terms of each of the other five.[1]

in terms of	$\sin\theta$	$\csc\theta$	$\cos\theta$	$\sec\theta$	$\tan\theta$	$\cot\theta$
$\sin\theta =$	$\sin\theta$	$\frac{1}{\csc\theta}$	$\pm\sqrt{1-\cos^2\theta}$	$\pm\frac{\sqrt{\sec^2\theta-1}}{\sec\theta}$	$\pm\frac{\tan\theta}{\sqrt{1+\tan^2\theta}}$	$\pm\frac{1}{\sqrt{1+\cot^2\theta}}$
$\csc\theta =$	$\frac{1}{\sin\theta}$	$\csc\theta$	$\pm\frac{1}{\sqrt{1-\cos^2\theta}}$	$\pm\frac{\sec\theta}{\sqrt{\sec^2\theta-1}}$	$\pm\frac{\sqrt{1+\tan^2\theta}}{\tan\theta}$	$\pm\sqrt{1+\cot^2\theta}$
$\cos\theta =$	$\pm\sqrt{1-\sin^2\theta}$	$\pm\frac{\sqrt{\csc^2\theta-1}}{\csc\theta}$	$\cos\theta$	$\frac{1}{\sec\theta}$	$\pm\frac{1}{\sqrt{1+\tan^2\theta}}$	$\pm\frac{\cot\theta}{\sqrt{1+\cot^2\theta}}$
$\sec\theta =$	$\pm\frac{1}{\sqrt{1-\sin^2\theta}}$	$\pm\frac{\csc\theta}{\sqrt{\csc^2\theta-1}}$	$\frac{1}{\cos\theta}$	$\sec\theta$	$\pm\sqrt{1+\tan^2\theta}$	$\pm\frac{\sqrt{1+\cot^2\theta}}{\cot\theta}$
$\tan\theta =$	$\pm\frac{\sin\theta}{\sqrt{1-\sin^2\theta}}$	$\pm\frac{1}{\sqrt{\csc^2\theta-1}}$	$\pm\frac{\sqrt{1-\cos^2\theta}}{\cos\theta}$	$\pm\sqrt{\sec^2\theta-1}$	$\tan\theta$	$\frac{1}{\cot\theta}$
$\cot\theta =$	$\pm\frac{\sqrt{1-\sin^2\theta}}{\sin\theta}$	$\pm\sqrt{\csc^2\theta-1}$	$\pm\frac{\cos\theta}{\sqrt{1-\cos^2\theta}}$	$\pm\frac{1}{\sqrt{\sec^2\theta-1}}$	$\frac{1}{\tan\theta}$	$\cot\theta$

Identity

$$\cos(\theta)\, x_i = \frac{1}{\sqrt{1+\tan^2(\theta)}}$$

- That means these:

$$x_f = \cos(\theta)\, x_i - \sin(\theta)\, y_i$$

$$y_f = \sin(\theta)\, x_i + \cos(\theta)\, y_i$$

- Can turn into these:

$$x_f = (x_i - \tan(\theta)\, y_i)\frac{1}{\sqrt{1+\tan^2(\theta)}}$$

$$y_f = (y_i + \tan(\theta)\, x_i)\frac{1}{\sqrt{1+\tan^2(\theta)}}$$

Let's ignore these

So now our task:

- Ignoring that factor on the outside does break stuff.
- We're no longer really doing a pure rotation
- …we have to call it something else…

The ' means value isn't same as before

$$x'_f = (x_i - \tan(\theta)\, y_i)$$

$$y'_f = (y_i + \tan(\theta)\, x_i)$$

We still don't know how to calculate $\tan(\theta)$*…that'll come*

PseudoRotations

- In a pseudorotation, you still rotate by the same angle, but you depart the unit circle:

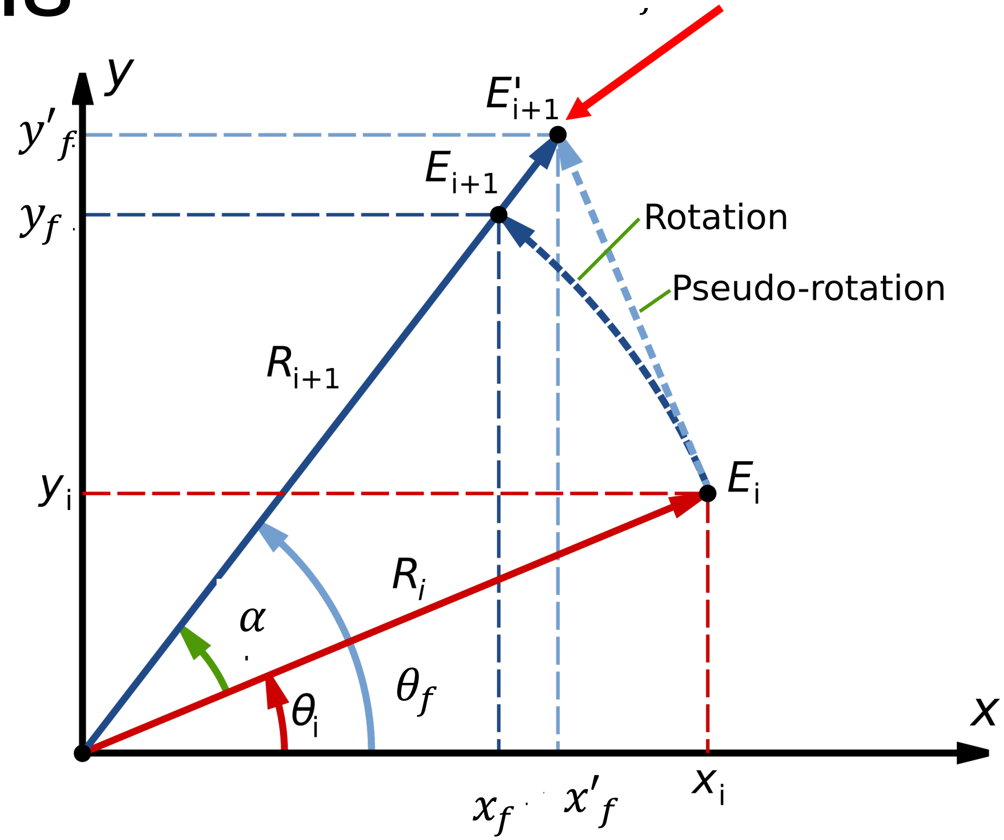
What we've got

$$x'_f = (x_i - \tan(\alpha)\, y_i)$$

$$y'_f = (y_i + \tan(\alpha)\, x_i)$$

Has length of

$$1 \cdot \sqrt{1 + \tan^2(\theta)}$$



What we wanted

$$x_f = (x_i - \tan(\alpha)\, y_i)\frac{1}{\sqrt{1 + \tan^2(\alpha)}} \qquad y_f = (y_i + \tan(\alpha)\, x_i)\frac{1}{\sqrt{1 + \tan^2(\alpha)}}$$

OK still though…

What we've got

$$x'_f = (x_i - \tan(\alpha)\, y_i)$$

$$y'_f = (y_i + \tan(\alpha)\, x_i)$$

- We still don't know $\tan(\theta)$
- Now we're using a thing we don't know, to do a thing we don't want….seems dumb if you ask me.

Iterations

- We don't have to do this move all at one time. We could do it in steps.
- Just like you can apply a matrix…then apply a matrix…you can do the same thing here.
- Do a bunch of smaller pseudo rotations forwards and even backwards (like a binary search)
- Since we know the angle we want, we could keep trac

step0

$$x_0 = (x_i - \tan(\alpha_0)\, y_i)$$

$$y_0 = (y_i + \tan(\alpha_0)\, x_i)$$

$$\theta_0 = 0 + \alpha_0$$

step1

$$x_1 = (x_0 - \tan(-\alpha_1)\, y_0)$$

$$y_1 = (y_0 + \tan(-\alpha_1)\, x_0)$$

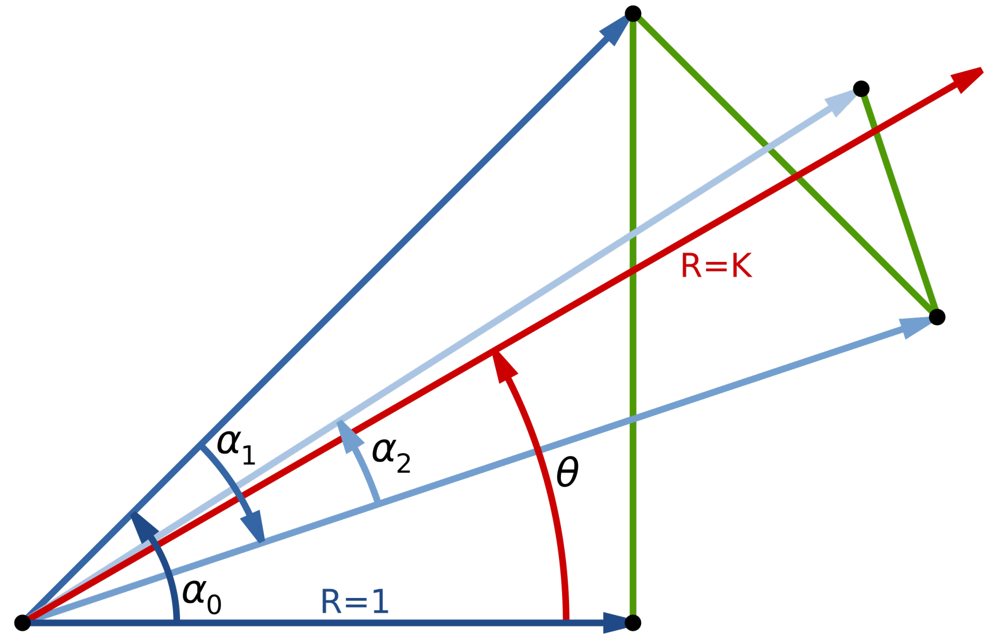
$$\theta_1 = 0 + \alpha_0 - \alpha_1$$

step2

$$x_2 = (x_1 - \tan(\alpha_2)\, y_1)$$

$$y_2 = (y_1 + \tan(\alpha_2)\, x_1)$$

$$\theta_2 = 0 + \alpha_0 - \alpha_1 + \alpha_2$$



stepn...

$$x_n = (x_{n-1} - \tan(\alpha_n)\, y_{n-1})$$

$$y_n = (y_{n-1} + \tan(\alpha_n)\, x_{n-1})$$

$$\theta_n = 0 + \alpha_0 - \alpha_1 + \alpha_2 + \cdots \alpha_n$$

OK interesting…

- If we could conceivably arrive at an arbitrary angle using a number of other steps…

- Could we pick a collection of steps that could be used to arrive at most arbitrary angles (within reason?)

- And could we pre-compute those angles?

If we have these precomputed angle jumps…

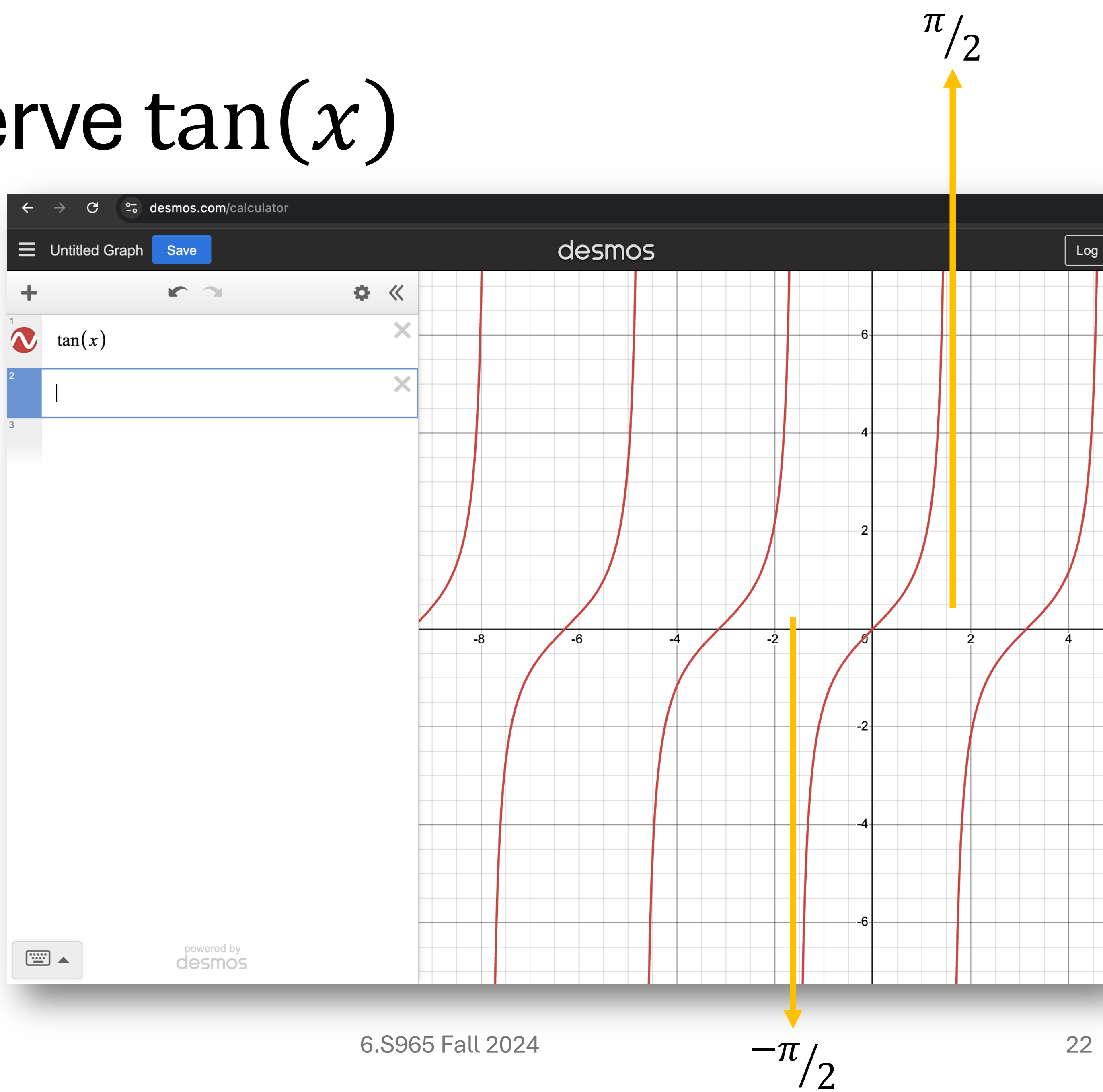
- Then we could potentially iterate towards our target θ with a number of pre-calculated α jumps

- We could keep track if our running tally is $>$ or $< \theta$ and add or subtract our α as needed.

What do we want in our precomputed α?

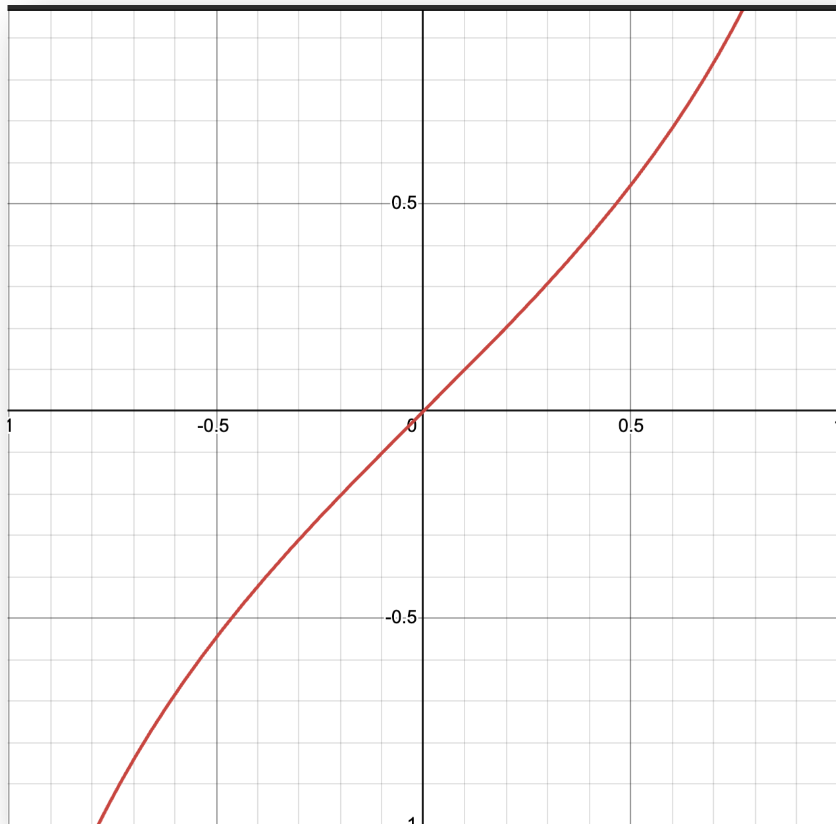
- Actually nothing.
- What we really care about are good, clean, wholesome, easy-to-apply values of $\tan(\alpha)$
- And remember we're not in human land, we're in digital land…so what are nice and easy to apply are in base 2!
- So are there any nice base-2
- And it sure would be nice to have angles that could go forwards or backwards

Observe $\tan(x)$



$\tan(x)$ is symmetric

- That's nice…that means we could just store precomputed values of $\tan(\alpha)$ for $\alpha > 0$ and just flip signs when needed.



Are there any "nice" $\tan(\alpha)$

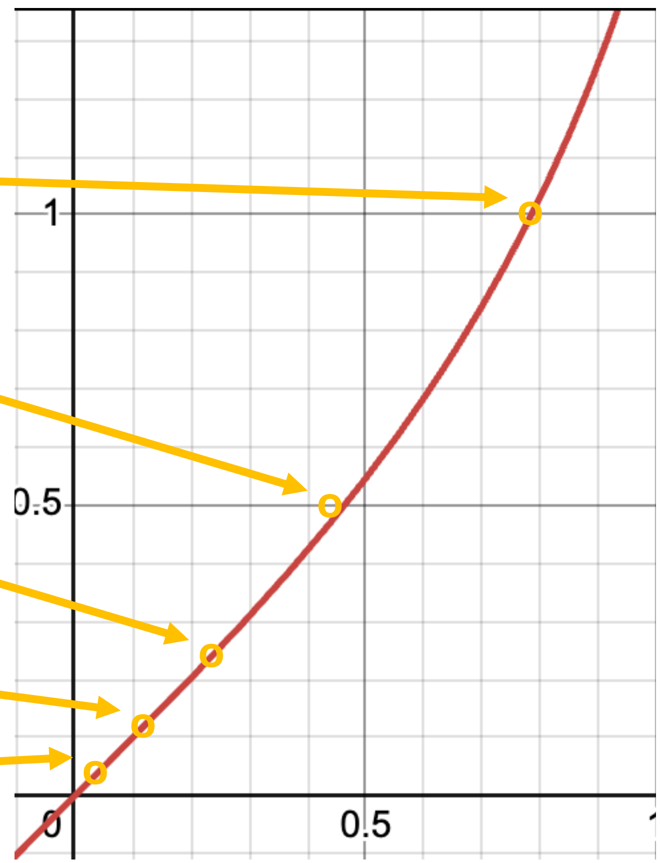
$\tan(\alpha) = 1 \ @\alpha = 45^o$

$\tan(\alpha) = \frac{1}{2} \ @\alpha = 26.57^o$

$\tan(\alpha) = \frac{1}{4} \ @\alpha = 14.04^o$

$\tan(\alpha) = \frac{1}{8} \ @\alpha = 7.13^o$

many more …
Get smaller and smaller



Do this for a bunch of power-of-2 values

- Can generate a whole table…basically as many as you want
- The only nasty thing you need to store would be these precomputed angles
- Because now all those multiplications by tangents are are easy.

i	$a_i = \tan^{-1}(2^{-i})$	
	Degrees	**Radians**
0	45.00	0.7854
1	26.57	0.4636
2	14.04	0.2450
3	7.13	0.1244
4	3.58	0.0624
5	1.79	0.0312
6	0.90	0.0160
7	0.45	0.0080
8	0.22	0.0040
9	0.11	0.0020

step0

$$x_0 = (x_i - \tan(\alpha_0)\, y_i)$$

$$y_0 = (y_i + \tan(\alpha_0)\, x_i)$$

$$\theta_0 = 0 + \alpha_0$$

step1

$$x_1 = (x_0 - \tan(-\alpha_1)\, y_0)$$

$$y_1 = (y_0 + \tan(-\alpha_1)\, x_0)$$

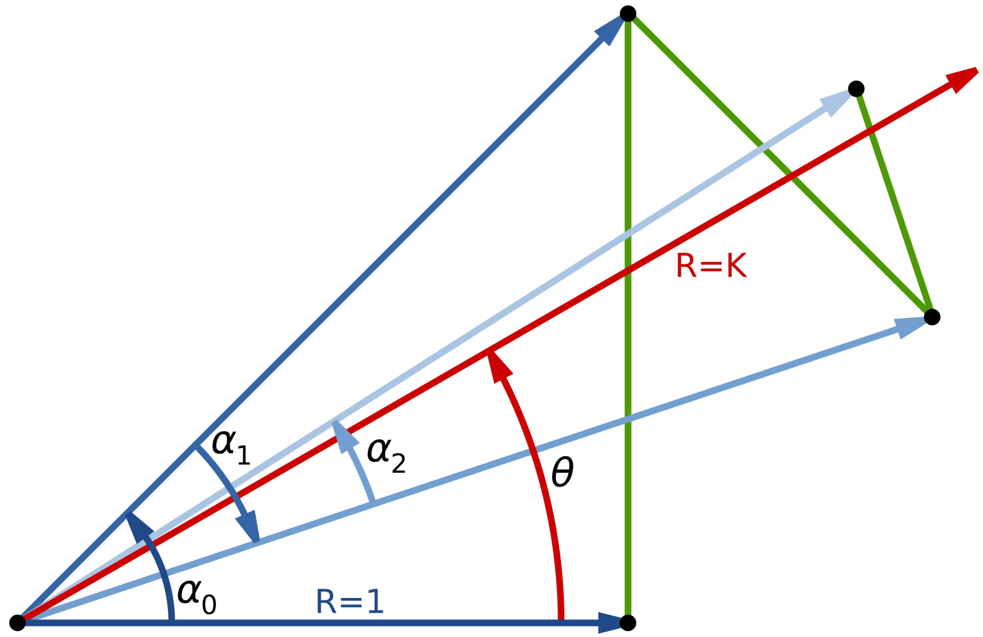
$$\theta_1 = 0 + \alpha_0 - \alpha_1$$

step2

$$x_2 = (x_1 - \tan(\alpha_2)\, y_1)$$

$$y_2 = (y_1 + \tan(\alpha_2)\, x_1)$$

$$\theta_2 = 0 + \alpha_0 - \alpha_1 + \alpha_2$$



stepn...

$$x_n = (x_{n-1} - \tan(\alpha_n)\, y_{n-1})$$

$$y_n = (y_{n-1} + \tan(\alpha_n)\, x_{n-1})$$

$$\theta_n = 0 + \alpha_0 - \alpha_1 + \alpha_2 + \cdots \alpha_n$$

step0

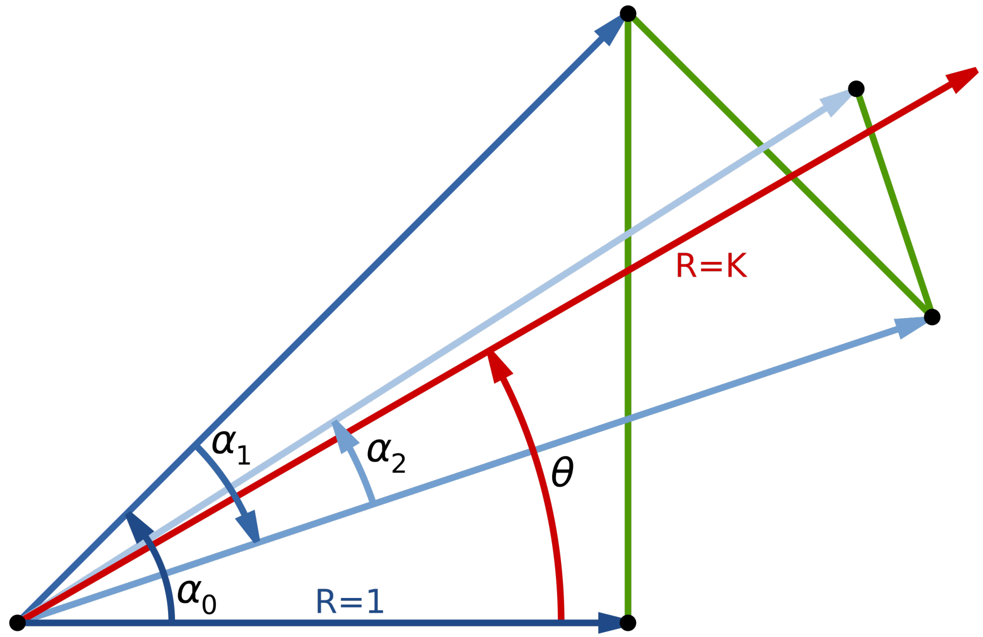
$$x_0 = (x_i - 1 \cdot y_i)$$
$$y_0 = (y_i + 1 \cdot x_i)$$
$$\theta_0 = 0 + 45$$

step1

$$x_1 = \left(x_0 - {}^{1}\!/_{2}\, y_0\right)$$
$$y_1 = \left(y_0 + {}^{1}\!/_{2}\, x_0\right)$$
$$\theta_1 = 0 + 45 - 26.57$$

step2

$$x_2 = \left(x_1 - {}^{1}\!/_{4}\, y_1\right)$$
$$y_2 = \left(y_1 + {}^{1}\!/_{4}\, x_1\right)$$
$$\theta_2 = 0 + 45 - 26.57 + 14.04$$



stepn...

$$x_n = \left(x_{n-1} - {}^{1}\!/_{2^n}\, y_{n-1}\right)$$
$$y_n = \left(y_{n-1} + {}^{1}\!/_{2^n}\, x_{n-1}\right)$$
$$\theta_n = 0 + 45 - 26.57 + 14.04 \ldots \alpha_n$$

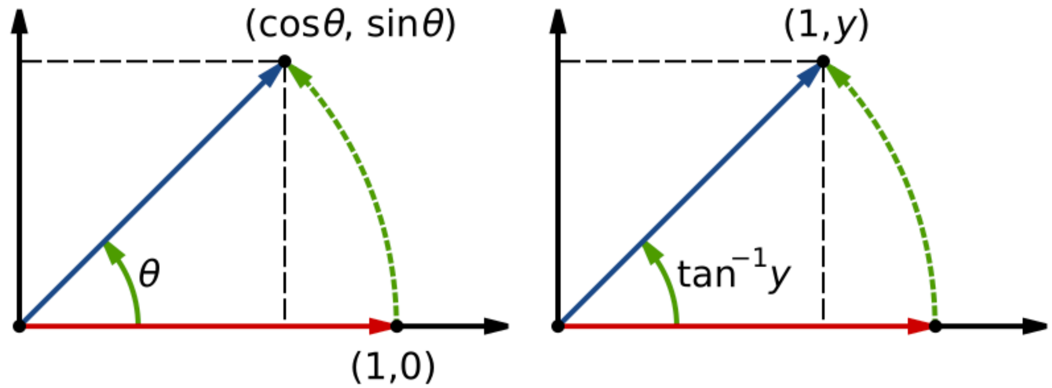
More and More

- The more iterations you do, the closer and closer you'll be able to get your final angle to your desired angle.
- It works out to about 1 bit of precision per iteration.

- But we're still not there yet.

We wanted to do this…

- Rotate things.



- Start at (1, 0)
- Rotate by θ
- We get $(\cos\theta, \sin\theta)$

- Start at $(1, y)$
- Rotate until $y = 0$
- The rotation is $\tan^{-1} y$

- But we're not…We're pseudo-rotating :/

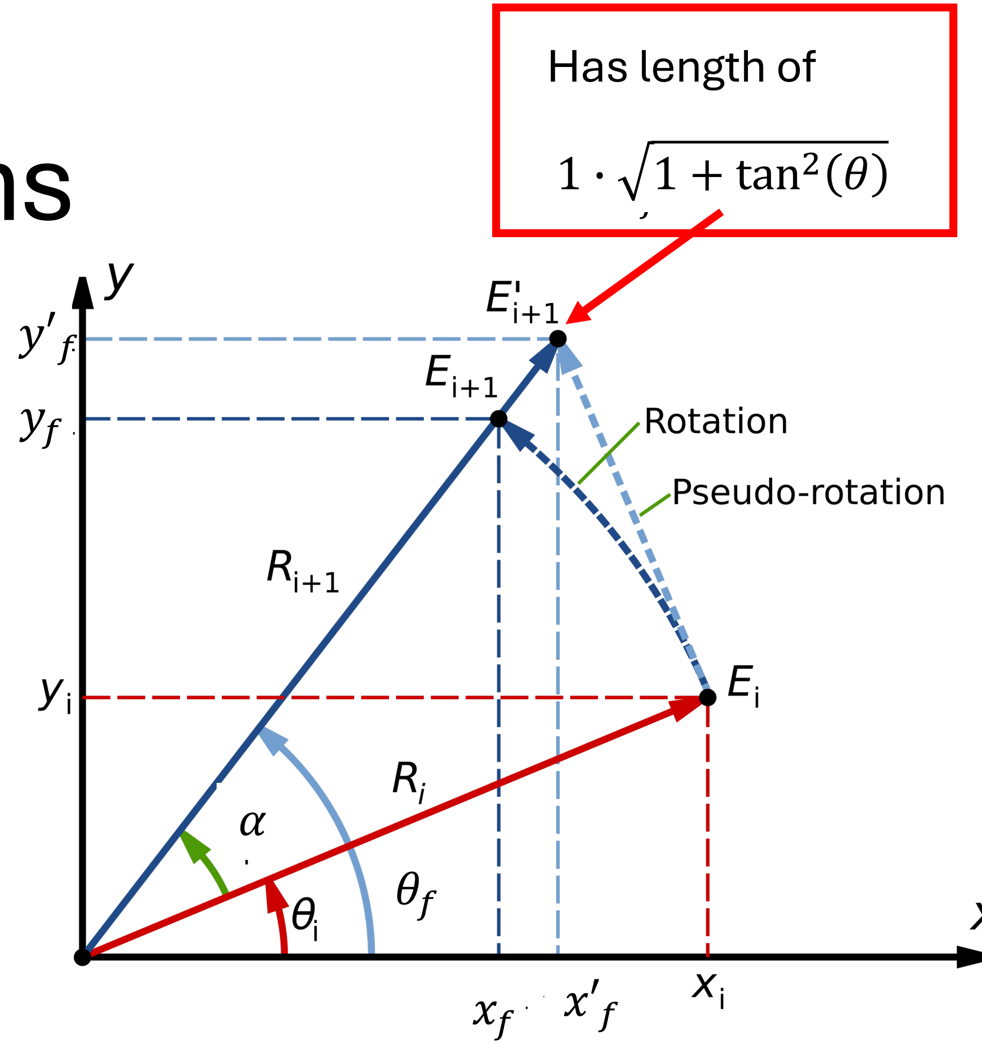
PseudoRotations

- In a pseudorotation, you still rotate by the same angle, but you depart the unit circle:

What we've got

$$x'_f = (x_i - \tan(\alpha)\, y_i)$$

$$y'_f = (y_i + \tan(\alpha)\, x_i)$$



What we wanted

$$x_f = (x_i - \tan(\alpha)\, y_i)\frac{1}{\sqrt{1+\tan^2(\alpha)}} \qquad y_f = (y_i + \tan(\alpha)\, x_i)\frac{1}{\sqrt{1+\tan^2(\alpha)}}$$

Remember...

$$\cos(\theta)\, x_i = \frac{1}{\sqrt{1+\tan^2(\theta)}}$$

- That means these:

$$x_f = \cos(\theta)\, x_i - \sin(\theta)\, y_i$$

$$y_f = \sin(\theta)\, x_i + \cos(\theta)\, y_i$$

- Can turn into these:

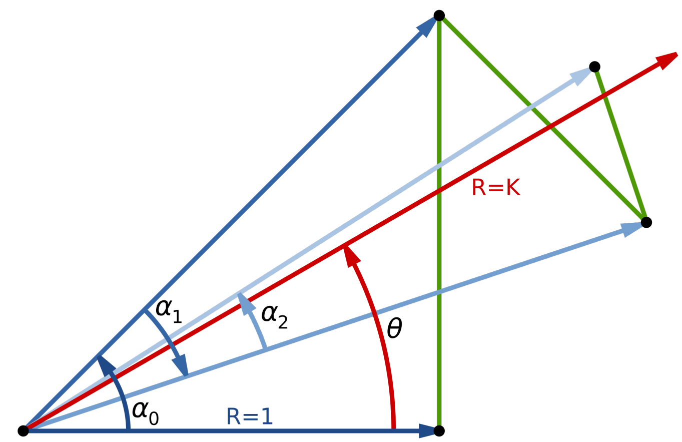
$$x_f = (x_i - \tan(\theta)\, y_i)\frac{1}{\sqrt{1+\tan^2(\theta)}}$$

$$y_f = (y_i + \tan(\theta)\, x_i)\frac{1}{\sqrt{1+\tan^2(\theta)}}$$

Let's ignore these

We can zero in on our angle…

- But the x,y final locations are still messed up
- On each iteration since we're not multiplying by $\frac{1}{\sqrt{1+\tan^2(\alpha_i)}}$…



- That means we're actually multiplying by $\sqrt{1+\tan^2(\alpha_i)}$
- You'll hear this called "gain" of a pseudorotation…

So after n iterations…

- I'd expect the vector to be this large…

$$K = \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \alpha_i}$$

- What will K be?

- It is going to depend on what/how we rotated right? And that is nasty…

BUTTTT!!!!

- We know ahead of time all those α values and because of their behavior around 0, it doesn't matter if we + or – with them
- For a given implementation…

$$K = \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \alpha_i}$$

- This will stay the same

i	$\alpha_i = \tan^{-1}(2^{-i})$	
	Degrees	**Radians**
0	45.00	0.7854
1	26.57	0.4636
2	14.04	0.2450
3	7.13	0.1244
4	3.58	0.0624
5	1.79	0.0312
6	0.90	0.0160
7	0.45	0.0080
8	0.22	0.0040
9	0.11	0.0020

Not only that

$$K = \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \alpha_i}$$

- All CORDIC implementations pick the same α values and these get smaller and smaller

- That means this product actually converges to a fixed value,
- which works out to be: 1.646760258121

i	$\alpha_i = \tan^{-1}(2^{-i})$	
	Degrees	**Radians**
0	45.00	0.7854
1	26.57	0.4636
2	14.04	0.2450
3	7.13	0.1244
4	3.58	0.0624
5	1.79	0.0312
6	0.90	0.0160
7	0.45	0.0080
8	0.22	0.0040
9	0.11	0.0020

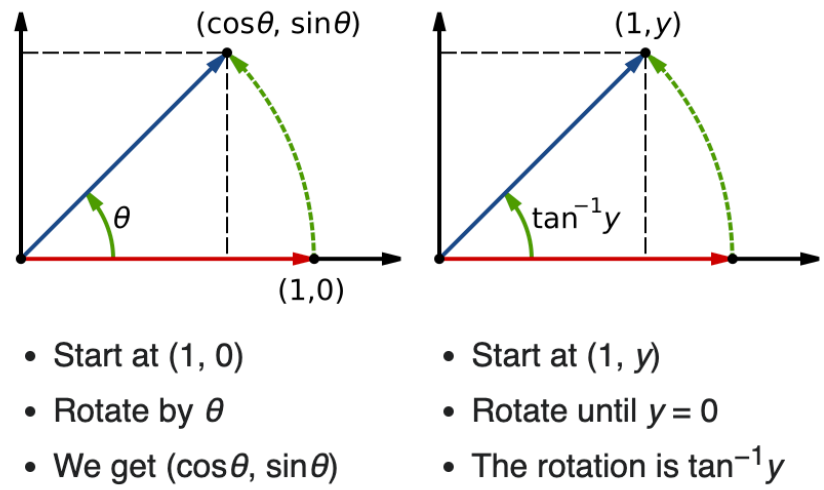
Smaller n smaller

So once you're done…

- You can take your x_f and y_f and multiplying by 0.60725293634
- Which is the same as multiplying by 2608131502 and right shifting by 32.

- You can also pre-multiply by this in your starting x_i and y_i

Generalizing CORDIC



- The pre-compute and step-by-step iterations are universal
- Their meaning and the target can be altered:
  - We previously targeted our accumulator to be θ
  - We could also target to get y to be 0…
    - The amount the accumulator ends up with is based on inverse tan of starting x and y
    - The amount x ends up with is based on the sqrt(x**2+y**2)

Generalized CORDIC

- The three equations we're iterating on can be generalized to this format

z is our angle accumulator

μ is settable constant

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i + d_i x_i 2^{-i}$$

d_i is our control/feedback function for locking into a target

$\mathrm{sgn}(\theta)$ in our walkthrough example

2^{-i} are the $\tan(\alpha_i)$ from our original example

Different Modes

Mode	Rotation	Vectoring
	$d_i = \operatorname{sgn}(z_i), \quad z \to 0$	$d_i = -\operatorname{sgn}(y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1} 2^{-i}$	$x, y, z \to$ CORDIC $\to$ $K(x\cos z - y\sin z)$, $K(y\cos z + x\sin z)$, 0	$x, y, z \to$ CORDIC $\to$ $K\sqrt{x^2+y^2}$, 0, $z+\tan^{-1}(y/x)$
Linear $\mu = 0$ $\alpha_i = 2^{-i}$	$x, y, z \to$ CORDIC $\to$ x, $y+xz$, 0	$x, y, z \to$ CORDIC $\to$ x, 0, $z+y/x$
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1} 2^{-i}$	$x, y, z \to$ CORDIC $\to$ $K'(x\cosh z - y\sinh z)$, $K'(y\cosh z + x\sinh z)$, 0	$x, y, z \to$ CORDIC $\to$ $K\sqrt{x^2-y^2}$, 0, $z+\tanh^{-1}(y/x)$

- In hyperbolic mode, iterations 4, 13, 40, 121, ..., j, $3j+1$,... must be repeated. The constant K' given below accounts for this.
- $K = 1.646760258121...$
- $1/K = 0.607252935009...$
- $K' = 0.8281593609602...$
- $1/K' = 1.207497067763...$

CORDIC

- You can use these outputs to generate all these weird things

Directly computable functions [edit | edit source]

$\sin z$	$\cos z$
$\tan^{-1} z$	$\sinh z$
$\cosh z$	$\tanh^{-1} z$
y/x	xz
$\tan^{-1}(y/x)$	$\sqrt{x^2+y^2}$
$\sqrt{x^2-y^2}$	$e^z = \sinh z + \cosh z$

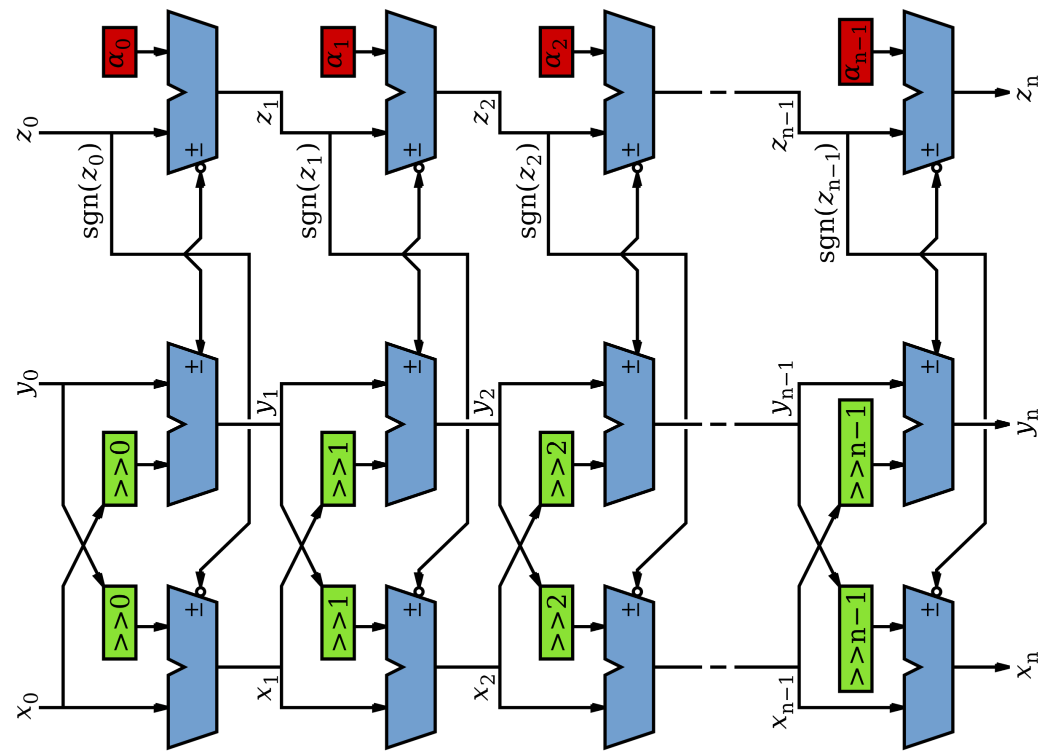
Indirectly computable functions [edit | edit source]

In addition to the above functions, a number of other functions can be produced by combining the results of previous computations:

$\tan z = \frac{\sin z}{\cos z}$	$\cos^{-1} w = \tan^{-1} \frac{\sqrt{1-w^2}}{w}$
$\tanh z = \frac{\sinh z}{\cosh z}$	$\sin^{-1} w = \tan^{-1} \frac{w}{\sqrt{1-w^2}}$
$\ln w = 2 \tanh^{-1} \frac{w-1}{w+1}$	$\log_b w = \frac{\ln w}{\ln b}$
$w^t = e^{t \ln w}$	$\cosh^{-1} = \ln\left(w + \sqrt{w^2-1}\right)$
$\tan^{-1}(y/x)$	$\sinh^{-1} = \ln\left(w + \sqrt{w^2+1}\right)$
$\sqrt{x^2-y^2}$	$\sqrt{w} = \sqrt{(w+1/4)^2 - (w-1/4)^2}$

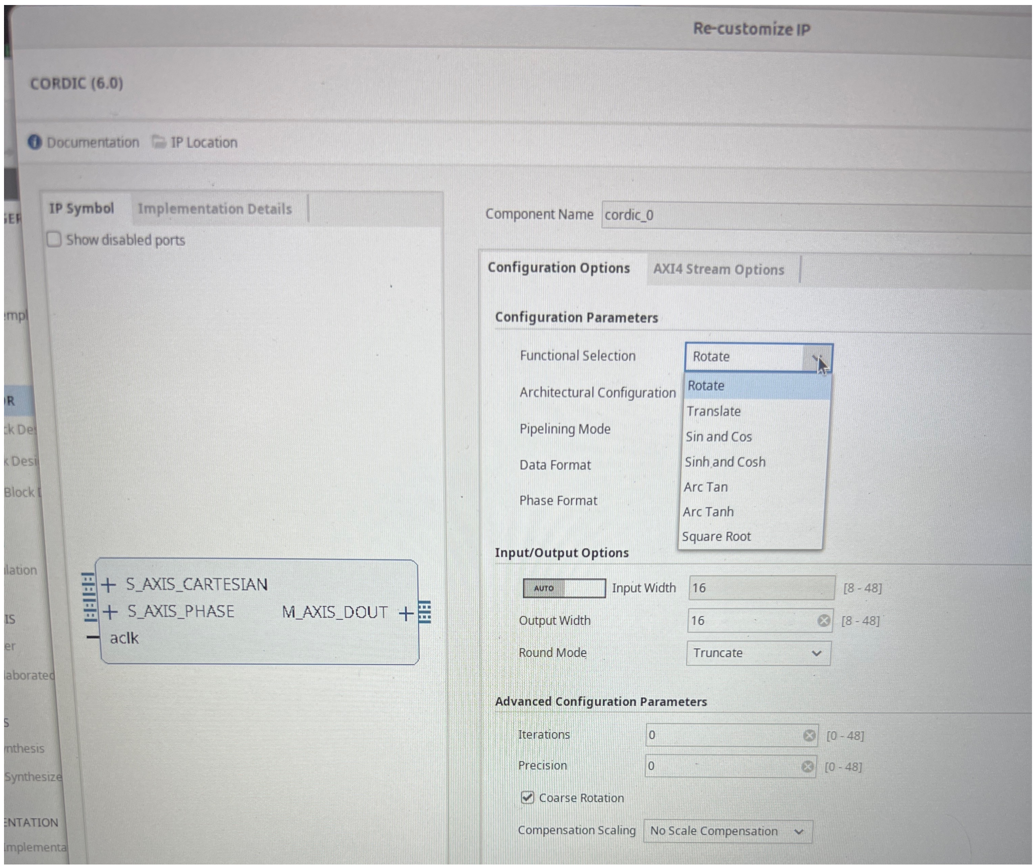
There’s very few multiplications in this

- And really no divisions.



https://en.wikibooks.org/wiki/Digital_Circuits/CORDIC

So when you're in Vivado or wherever



- Now you can know what this is doing and make a better one.

People still making improvements/updates

Generalized Hyperbolic CORDIC and Its Logarithmic and Exponential Computation With Arbitrary Fixed Base

Yuanyong Luo, Yuxuan Wang, Yajun Ha, *Senior Member, IEEE*, Zhongfeng Wang, *Fellow, IEEE*, Siyuan Chen, and Hongbing Pan

***Abstract*—This paper proposes a generalized hyperbolic COordinate Rotation Digital Computer (GH CORDIC) to directly compute logarithms and exponentials with an arbitrary fixed base. In a hardware implementation, it is more efficient than the state of the art which requires both a hyperbolic CORDIC and a constant multiplier. More specifically, we develop the theory of GH CORDIC by adding a new parameter called base to the conventional hyperbolic CORDIC. This new parameter can be used to specify the base with respect to the computation of logarithms and exponentials. As a result, the constant multiplier is no longer needed to convert base *e* (Euler's number) to other values because the base of GH CORDIC is adjustable. The proposed methodology is first validated using MATLAB with extensive vector matching. Then, example circuits with 16-bit fixed-point data are implemented under the TSMC 40-nm CMOS technology. Hardware experiment shows that at the highest frequency of the state of the art, the proposed methodology saves 27.98% area, 50.69% power consumption, and 6.67% latency when calculating logarithms; it saves 13.09% area, 40.05% power consumption, and 6.67% latency when computing exponentials. Both calculations do not compromise accuracy. Moreover, it can increase 13% maximum frequency and reduce up to 17.65% latency accordingly compared to the state of the art.**

***Index Terms*—Architecture, exponential, generalized hyperbolic COordinate Rotation Digital Computer (GH CORDIC),**

evaluate logarithms and exponentials: approximation method and iterative method. Although loads of well-related research achievements have been proposed on these methods, there is still plenty of room for improvement. First, current approaches do not support easy porting to other fixed bases while they are needed. Second, current approaches still have room to further reduce the hardware overheads. In this paper, we will propose a promising solution to abovementioned concerns.

The following literature addresses the evaluation of logarithms and exponentials using the approximation method. [1]–[4] evaluate binary logarithms and exponentials via simple piecewise linear approximation. When the output approaches zero, this method encounters notably large relative error. In order to overcome this shortage, Nam *et al.* [5] perform finer subdivisions around the output of zero since the error increases as the output value gets closer to zero. Subsequently, they have designed a processor of the logarithmic number system for 3-D graphics. The main shortcoming of a simple linear approximation method is the high relative error with limited lookup tables. Paul *et al.* [6] use a second-order polynomial approximation method to reduce the relative error. The

For Week 4

- I tell you to do a binary search for the square root, but actually a CORDIC would be cooler, tbh. You should try to get that working instead if you want.

- It should use lower resources and be better in general and cooler

Other Fun, Cheapies

- In lab this week, we we're finding essentially the magnitude of a complex number

- While the binary search/CORDIC will be better in terms of final value…there are others

- Alpha-Max-Plus-Beta-Min algorithm

Alpha Max Plus Beta Min

- Pretty cool approximation and some neat improvements with it

https://en.wikipedia.org/wiki/Alpha_max_plus_beta_min_algorithm