Fixed-point Trigonometric Functions for FPGAs

http://flopoco.gforge.inria.fr/

Matei Iştoan, Florent de Dinechin
Socrate, CITI lab – INRIA

Hardware Architectures for Evaluating Trigonometric Functions

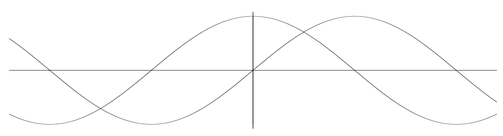
Why study trigonometric functions?

- fundamental in signal processing
  - FFT, modulation/demodulation, frequency synthesizers
- multitude of methods to compute the sine and/or cosine
  - best method for FPGAs?

Bit level complexity of computing sine/cosine?

Sine,cosine, or both?

- many applications require both
- some algorithms compute both



Three methods for computing sine and cosine

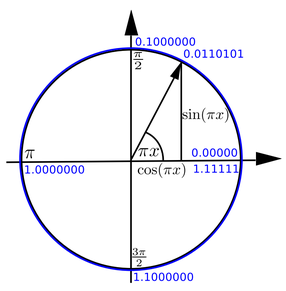
1. classical **CORDIC** algorithm
   suited for all classes of FPGAs
2. more specialized using **Taylor approximations** and **trigonometric identities**
   - for modern FPGAs, with **memories** and **multipliers**
3. generic polynomial approximation

Rules of the game

Fixed-Point Format Used

Input in $x \in [0, 1)$
$\longrightarrow$ compute $\sin(\pi x)$ and $\cos(\pi x)$

- 2π periodicity
- fixed-point periodicity



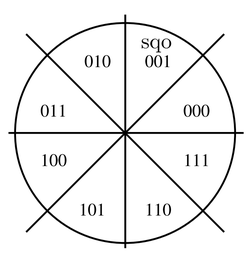
Scaled output: scale factor $1 - 2^{-w}$

- range not symmetric, 1 is not representable
- compute $\sin(\pi x) \times (1 - 2^{-w})$ and $\cos(\pi x) \times (1 - 2^{-w})$
- no penalty paid for the $(1 - 2^{-w})$ scaling factor

Goal: last bit accuracy

Argument Reduction

- based on the 3 MSBs of the input angle x
  - s - **s**ign
  - q - **q**uadrant
  - o - **o**ctant
- remaining argument $y' \in [0, 1/4)$

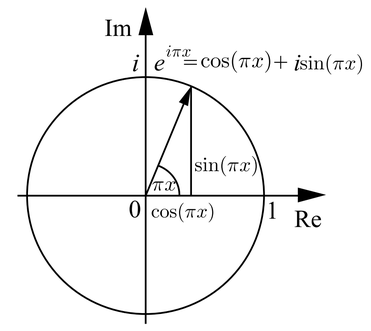


sqo	Reconstruction
000	$\begin{cases} \sin(\pi x) = \sin(\pi y') \\ \cos(\pi x) = \cos(\pi y') \end{cases}$
001	$\begin{cases} \sin(\pi x) = \cos(\pi y') \\ \cos(\pi x) = \sin(\pi y') \end{cases}$
010	$\begin{cases} \sin(\pi x) = \cos(\pi y') \\ \cos(\pi x) = -\sin(\pi y') \end{cases}$
011	$\begin{cases} \sin(\pi x) = \sin(\pi y') \\ \cos(\pi x) = -\cos(\pi y') \end{cases}$

Common Trigonometric Identities

Decompose a rotation in smaller sub-rotations:

$$e^{j(a+b)} = e^{ja} \times e^{jb}$$



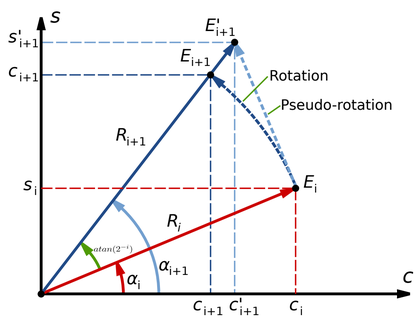
$$\begin{cases} \sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b) \\ \cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b) \end{cases}$$

CORDIC Method

The CORDIC Algorithm

Decomposition of the angle into micro-rotations

$$\begin{cases} c_0 = \frac{1}{\prod_{i=1}^{n}\sqrt{1+2^{-i}}} \\ s_0 = 0 \\ \alpha_0 = y \quad \text{(the reduced argument)} \end{cases}$$



$$\begin{cases} d_i = +1 \text{ if } \alpha_i > 0, \text{ otherwise } -1 \\ c_{i+1} = c_i - 2^{-i} d_i s_i \\ s_{i+1} = s_i + 2^{-i} d_i c_i \\ \alpha_{i+1} = \alpha_i - d_i \arctan(2^{-i}) \end{cases}$$

$$\begin{cases} c_{n\to\inf} = \cos(y) \\ s_{n\to\inf} = \sin(y) \\ \alpha_{n\to\inf} = 0 \end{cases}$$

Improvements

Reduced α-Datapath

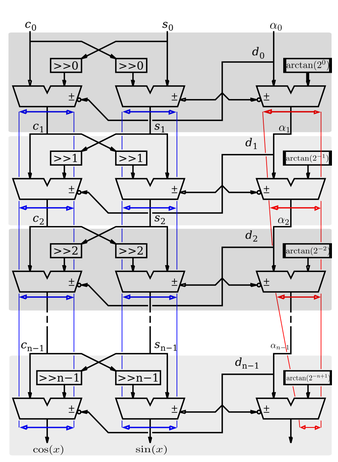
- 1 more MSB unused per iteration
- reduce the α-datapath by 1 bit per iteration
- saves space+latency

Reduced Iterations

- stop iterations earlier
  $\rightarrow$ replaced by a single rotation

$$\begin{cases} x_{i+1} = x_i + \alpha y_i \\ y_{i+1} = y_i - \alpha x_i \end{cases}$$

- only 2 multiplications are needed



Error Analysis: insure last-bit accuracy

$$\varepsilon_{method} + \varepsilon_{round} + \varepsilon_{finalrounding} < 2^{-w}$$

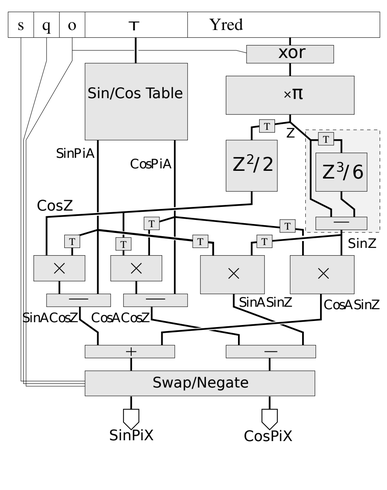
- method error (ε_{method}) $\longrightarrow$ number of iterations: smallest i for which $\varepsilon_{method} < 2^{-w-2}$
- rounding errors (ε_{round}) **on the α datapath**
  - total error on the α-datapath: $nb\_iter \times 2^{-w-g}$
- rounding errors **on the** sin() **and** cos() **datapath**
  - larger than the error on the α-datapath
- final rounding of sin and cos has error of 2^{-w-1}

Ad-hoc Method Based on Tables and Multiplications

Method Description

- angle split: y (*the reduced angle*) $= t + y_{red}$
  - t on a bits and y_{red} such that $y_{red} < 2^{-(a+2)}$
- store $\sin(\pi t)$ and $\cos(\pi t)$ in tables
- evaluate $\sin(\pi y_{red})$ and $\cos(\pi y_{red})$ using a Taylor polynomial approximation
  - need to compute first $z = y_{red} \times \pi$
  - $\sin(z) = z - z^3/6$ and $\cos(z) = 1 - z^2/2$
- reconstruct the values of $\sin(\pi y)$ and $\cos(\pi y)$ using

$$\begin{cases} \sin(\pi(t + y_{red})) = \sin(\pi t)\cos(\pi y_{red}) + \cos(\pi t)\sin(\pi y_{red}) \\ \cos(\pi(t + y_{red})) = \cos(\pi t)\cos(\pi y_{red}) - \sin(\pi t)\sin(\pi y_{red}) \end{cases}$$



Error Analysis

- $\frac{1}{2}ulp$ lost per table
- $1\,ulp$ per truncation and truncated multiplier/squarer
- $1\,ulp$ for computing $\frac{1}{4} - y$ (as $\neg y$)
- total of $15\,ulps$, independent of the input width
- $\rightarrow$ gives **g=4**

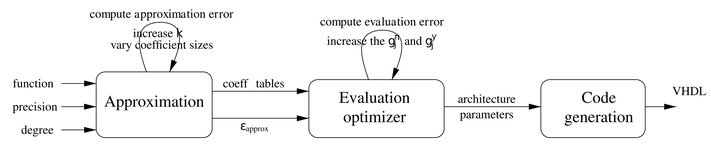
Shown in the results tables as SinAndCos.

Polynomial Evaluator-based Method

Polynomial Evaluator-based method

- computes only one of the functions, depending on an input
- uses an existing, generic polynomial approximation architecture generator

Shown in the results tables as SinOrCos.



Results

Approach	precision	latency	frequency	Reg. + LUTs	BRAM	DSP
CORDIC	16bit	18	478	969 + 1131	0	0
Red. CORDIC		13	368	625 + 719	0	2
SinAndCos		4	298	107 + 297	0	5
SinOrCos (d=2)		9	251	136 + 183	1	2
CORDIC	32bit	37	403.5	3495 + 3591	0	0
Red. CORDIC		24	256.8	2160 + 2234	0	4
SinAndCos		10	253	535 + 789	3	9
SinOrCos (d=3)		14	251	444 + 536	4	5

Synthesis results for Virtex6 FPGAs

- **Open Source** software with **state of the art** implementations
- **Fair comparison** of solutions
- CORDIC method **on par with vendor-provided solutions**
- CORDIC not viable for DSP-less FPGAs
- **Table and multiplications based solution best overall**

Approach	latency	area
CORDIC 16 bits	30.3 ns	1034 LUTs
SinAndCos 16 bits	15.0 ns	1211 LUTs
CORDIC 24 bits	44.6 ns	2079 LUTs
SinAndCos 24 bits	17.0 ns	2183 LUTs
CORDIC 32 bits	62.1 ns	3513 LUTs
SinAndCos 32 bits	19.4 ns	3539 LUTs

Logic only implementations

Approach	precision	latency	frequency	Reg. + LUTs	BRAM	DSP
CORDIC	40bit	45	375	5070 + 5289	0	0
Red. CORDIC		37	252	3695 + 3768	0	8
SinAndCos (bit heap)		11	266	895 + 1644	3	12
SinAndCos (table $z^3/6$)		8	232	500 + 949	4	12
SinOrCos (d=3)		15	251	628 +725	4	8
SinAndCos (bit heap)	48bit	13	232	1322 + 2369	12	17
SinOrCos		15	250	734 + 879	17	10

Synthesis results for Virtex6 FPGAs