Faithful Single-Precision Floating-Point Tangent for FPGAs

Martin Langhammer and Bogdan Pasca

ALTERA

Feb 12, 2013
What?

Tangent function:

- **periodic**, input range restricted to \((-\pi/2, +\pi/2)\)
- **symmetrical** to the origin: \(\tan(-x) = -\tan(x)\)
- Taylor series:

\[
\tan(x) = x + \frac{1}{3} x^3 + \frac{2}{15} x^5 + \ldots \quad x \in \left(\frac{-\pi}{2}, \frac{\pi}{2}\right)
\]
Compute in **floating-point** (IEEE-754)

Triplet *(sign, exponent, fraction)* defines $x$:

$$x = (-1)^s 2^e 1.f$$

Focus on **single-precision** $w_E = 8$ (exp. width), $w_F = 32$ (frac. width)

Perform **faithful rounding**: 

- **FR** - faithful rounding
- **CR** - correct rounding
- **ulp** - unit in the last place

floating-point numbers
Restrict input to fixed-point

\[ \tan(x) \approx x \text{ for } x < 2^{-w_F/2} \]

dynamic input range: \([2^{-w_F/2}, +\pi/2]\)

input in error-free fixed-point on \(1 + w_F + \lceil w_F/2 \rceil\) bits
(24+12=36 bits for single precision).

Use mathematical identities:

\[
\tan(a + b) = \frac{\tan(a) + \tan(b)}{1 - \tan(a)\tan(b)},
\]

\[
\tan(a + b + c) = \frac{\tan(a) + \tan(b) + \tan(c)}{1 - \tan(a)\tan(b) - \tan(a)\tan(b)\tan(c)}.
\]
How? - Single-precision specific simplifications

- use the **fixed-point decomposition** of the input argument
  
  \[
  \begin{align*}
  c & \text{- 9bit} & a & \text{- 9bit} & b & \text{- 18bit}
  \end{align*}
  \]

- **simplify:**
  - \(\tan(a)\) and \(\tan(b)\) small \(\rightarrow\) \(\tan(a)\tan(b)\) very small
  - \(b < 2^{-17}\) safe to use \(\tan(b) \approx b\)

  \(\rightarrow\) tangent computed using:

  \[
  \tan(x) = \frac{\tan(c) + \tan(a) + b}{1 - (\tan(a) + b)\tan(c)}
  \]
How? - Faithful precision requirement

\[ E_{\text{total}} = E_{\text{approx}} + E_{\text{round}} \]

- \( E_{\text{round}} \) pack result to floating-point (nearest, \( 1/2 \text{ulp} \))
- \( E_{\text{approx}} \) method errors + datapath trimmings
- tangent implemented as FP multiplication

\[ p = n \times id \]

- target: keep \( E_{\text{approx}} < 1/2 \text{ulp} \)

... some steps later:
→ for single-precision \( p = 24 \) (error bound slightly better than \( 1/4 \text{ulp} \) for numerator and inverse denominator)
How? - Datapath width

- **certify** approximations for the *numerator*:
  1. $\tan(c) = 0$ and $\tan(a)\tan(b)$ maximal:
     - $a = .\ 111111111$
     - $b = .\ 111111111111111000$
     - relative error is slightly less than $2^{-25}$, and should be $2^{-26}$.
     - but denominator is 1 and carries no error $\rightarrow$ accuracy reached
  2. $\tan(c)$ minimal but $> 0$ and $\tan(a)\tan(b)$ maximal
     - $\tan(a) < \tan(c)$ relative error is $2^{-26}$ (tabulated precision for $\tan(c)$)
     - compute both $\tan(a)$ and $\tan(b)$ with $1 + w_F + 2$ bits of accuracy.

- **certify** approximations for *denominator*:
  - possible cancellation amplifies existing errors
  - avoid large cancellation using additional table
  - tabulate results for $256ulp$ before $\pi/2$
  - largest cancellation can now be produced by:
    - $c = 1.10010010$
    - $a = .\ 000111001$
    - $b = .\ 010000$
  - cancellation size is 3 bits $\rightarrow$ 3 additional bits for right term
  - compute $\tan(a)$ and $\tan(c)$ on $1 + w_F + 2 + 3$ bits with $0.5ulp$ of accuracy.
- Normalize product denominator
- \( \tan(1.57 - 0.002) \)
- LUT \( \tan(x) \)
- 256ulp rounded to \( \pi/2 \)
- Bias exponent \( X'1' \)
- Fixed-point LUT \( \tan(0.002 - 7 \cdot 10^{-6}) \)
- [26:18] exponent numerator
- Fraction exponent \( [17:0] \)
- [35:27] friction
- FP tan(c)
- \( a + b \)
- LUT tan(a + b)
- FP tan(c) + \( b[17:0] \)
- [15] numerator fraction
- [17:0] denominator fraction
- LZC numerate numerator fraction
- Normalize numerator denominator
- 1
- 1/x numerator fraction
- LUT tan(x) numerator fraction
- LUT tan(x) denominator fraction
- Rounding Exception Handling
- \( \pi/2 - 256ulp \) to \( \pi/2 \)
## Results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Lat @ Freq.</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours</td>
<td>30 @ 314MHz</td>
<td>18MUL, 8M9K, 1172LUT, 1078Reg</td>
</tr>
<tr>
<td>tan((\pi x)) [1]</td>
<td>48 @ 360MHz</td>
<td>28MUL, 7M9K, 2633LUT, 4099Reg</td>
</tr>
<tr>
<td>sin cos((\pi x)) [2]</td>
<td>85ns</td>
<td>10 MUL, 2*1365 LUTs</td>
</tr>
<tr>
<td>div [3]</td>
<td>16 @ 233MHz</td>
<td>1210LUT, 1308REG</td>
</tr>
<tr>
<td>div [4]</td>
<td>11 @ 400MHz</td>
<td>8MUL, 4M9K, 274LUT, 291Reg</td>
</tr>
</tbody>
</table>

- shorter latency
- fewer resources


Conclusion

- we implement the tangent function as a **fused operator**
- exploit FPGA flexibility: **exotic formats**, fixed-point and floating-point
- careful error analysis → **compute just right**
- make **efficient use of** existing **FPGA resources**
  (memories and multipliers)
Thank you and see you at the poster!