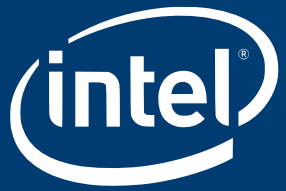


# SP FP TANGENT ARCHITECTURE FOR FPGAS

Bogdan Pasca, Martin Langhammer
Intel PSG

Arithmetic in DSP (Special Session), Tuesday, July 25th, 2017

# This talk is about

- single-precision tangent implementation
- input range restricted to $[-\pi/2, \pi/2]$
- accuracy can be tunned to meet OpenCL conformance
- target contemporary FPGAs having hard FP DSPs
- available in DSP Builder Advanced backend

- this work builds on
  Faithful Single-Precision Floating-Point Tangent for FPGAs
  Field Programmable Gate Arrays 2013 (FPGA'13)

# Context

- many trigonometric function implementation use CORDIC [8, 3]
  - efficient for iterative implementations
  - unrolled implementations stressful on FPGA fabric
  - multiple, deep, arithmetic structures with wide adders

# Context

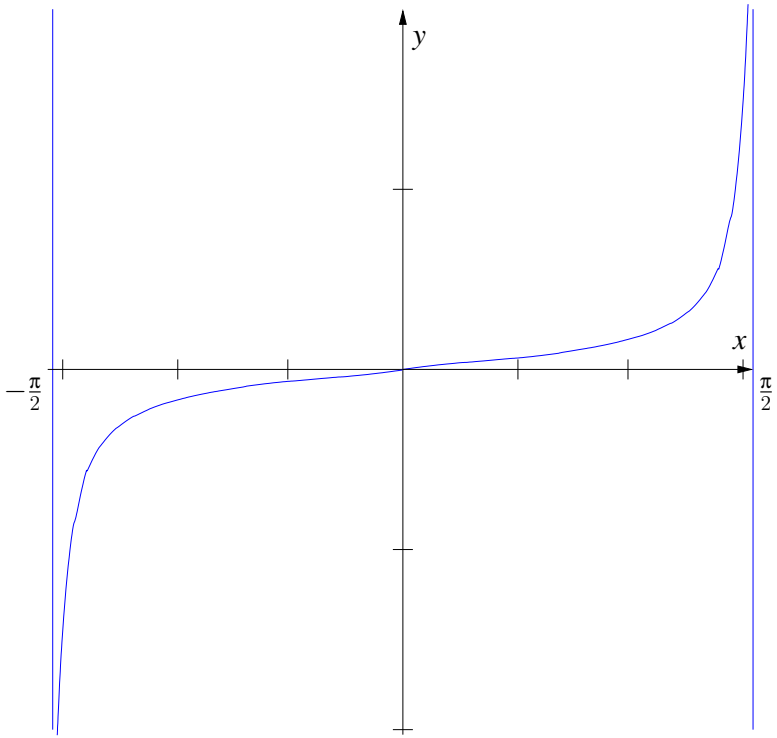
- many trigonometric function implementation use CORDIC [8, 3]
  - efficient for iterative implementations
  - unrolled implementations stressful on FPGA fabric
  - multiple, deep, arithmetic structures with wide adders

- piecewise polynomial-approximation can also be used [6]
  - better mapping to FPGA architecture
  - can use operator assembly, but that can be wasteful [1, 5]

# Context

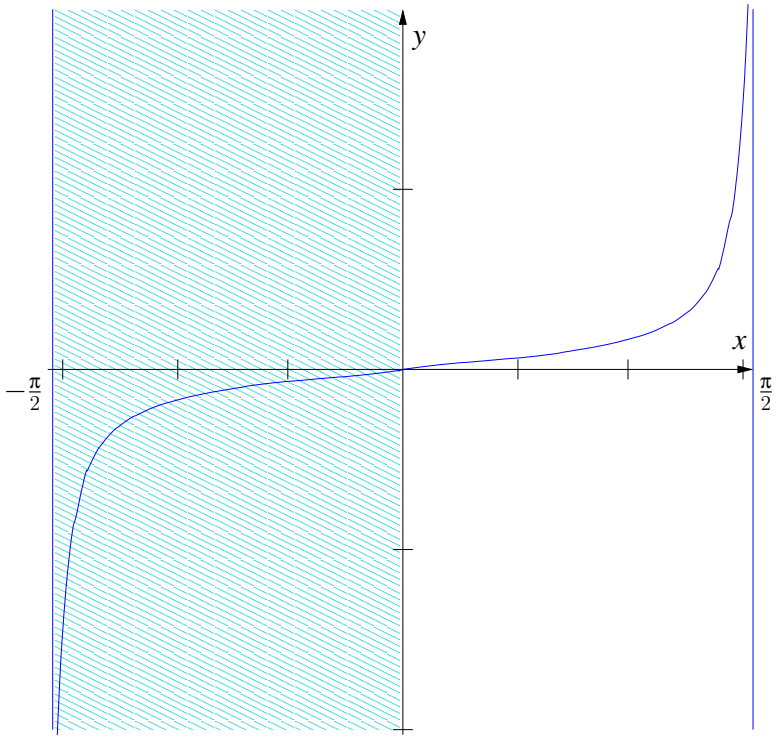
- many trigonometric function implementation use CORDIC [8, 3]
  - efficient for iterative implementations
  - unrolled implementations stressful on FPGA fabric
  - multiple, deep, arithmetic structures with wide adders
- piecewise polynomial-approximation can also be used [6]
  - better mapping to FPGA architecture
  - can use operator assembly, but that can be wasteful [1, 5]
- implementation as a fused-operator
  - careful error analysis
  - can obtain faithfully rounded SP tangent

# Technique



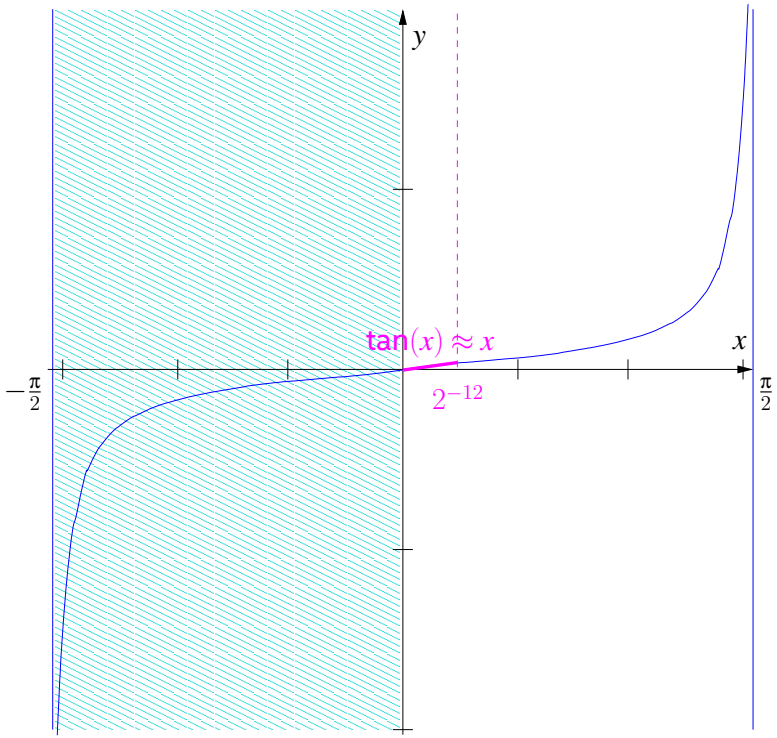

- compute $\tan(x)$ with $x \in [-\pi/2, +\pi/2]$
- range-reduction step not presented here [2, 7]

# Technique




- compute $\tan(x)$ with $x \in [-\pi/2, +\pi/2]$
- range-reduction step not presented here [2, 7]
- tangent is symmetrical: $\tan(-x) = -\tan(x) \rightarrow$ range $= [0, \pi/2)$.

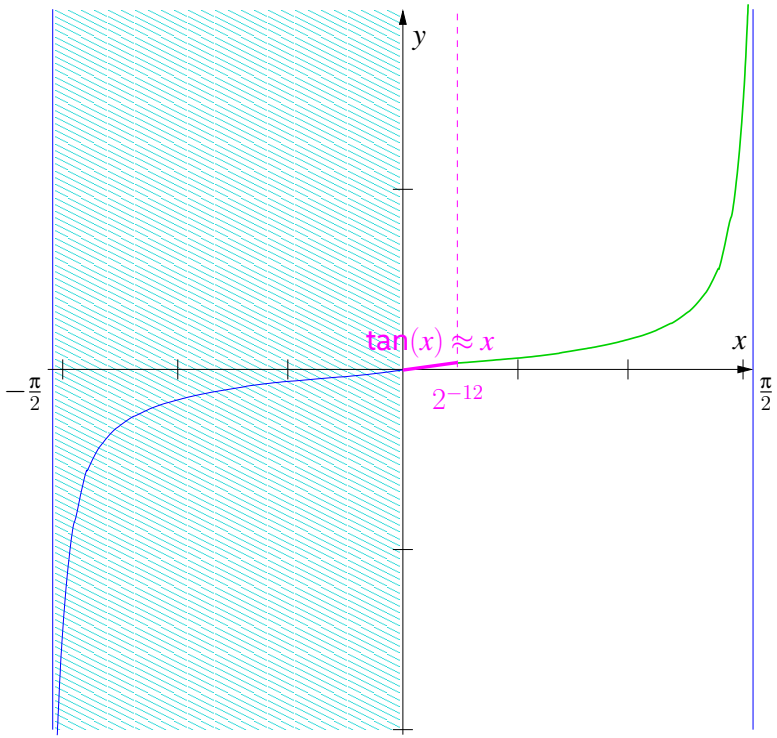
# Technique




- compute $\tan(x)$ with $x \in [-\pi/2, +\pi/2]$
- range-reduction step not presented here [2, 7]
- tangent is symmetrical: $\tan(-x) = -\tan(x) \rightarrow$ range $= [0, \pi/2)$.
- $x$ is very small ($< 2^{-12}$ in SP), good approximation is $\tan(x) = x$

$$\tan(x) = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \ldots$$

# Technique




- compute $\tan(x)$ with $x \in [-\pi/2, +\pi/2]$
- range-reduction step not presented here [2, 7]
- tangent is symmetrical: $\tan(-x) = -\tan(x) \rightarrow$ range $= [0, \pi/2)$.
- $x$ is very small ($< 2^{-12}$ in SP), good approximation is $\tan(x) = x$

$$\tan(x) = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \ldots$$

- 24+12=36 bit fixed-point for compute range input

# Technique

- the following identity holds:

$$\tan(a+b) = \frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}$$

# Technique

- the following identity holds:

$$\tan(a+b) = \frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}$$

- this can be expanded recursively to:

$$\tan(a+b+c) = \frac{\frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}+\tan(c)}{1-\frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}\tan(c)}$$

- decompose input as: $x = c+a+b$
  - weight of the MSB of $c$ is $2^0$

$x$

| c | a | b |
|---|---|---|
| 9 bits | 9 bits | 18 bits |

# Approximating the numerator

$$n = \frac{\tan(a) + \tan(b)}{1 - \tan(a)\tan(b)} + \tan(c)$$

- $\tan(b) \in [0, 2^{-17}]$, $\tan(a) \in [0, 2^{-8}]$
- $\tan(a)\tan(b) \in [0, 2^{-25}] \rightarrow 1 - \tan(a)\tan(b) \approx 1$

# Approximating the numerator

$$n = \frac{\tan(a) + \tan(b)}{1 - \tan(a)\tan(b)} + \tan(c)$$

- $\tan(b) \in [0, 2^{-17}]$, $\tan(a) \in [0, 2^{-8}]$
- $\tan(a)\tan(b) \in [0, 2^{-25}] \rightarrow 1 - \tan(a)\tan(b) \approx 1$

- $b < 2^{-17}$ good approximation is $\tan(b) \approx b$.

# Approximating the numerator

$$n = \frac{\tan(a) + \tan(b)}{1 - \tan(a)\tan(b)} + \tan(c)$$

- $\tan(b) \in [0, 2^{-17}]$, $\tan(a) \in [0, 2^{-8}]$
- $\tan(a)\tan(b) \in [0, 2^{-25}] \rightarrow 1 - \tan(a)\tan(b) \approx 1$
- $b < 2^{-17}$ good approximation is $\tan(b) \approx b$.
- numerator approximated by:

$$n = \tan(a) + b + \tan(c)$$

# Approximating the denominator

$$d = 1 - \frac{\tan(a) + \tan(b)}{1 - \tan(a)\tan(b)} \tan(c)$$

- use the same technique $1 - \tan(a)\tan(b) \approx 1$, $\tan(b) \approx b$
- possible cancellation can amplify an existing error
- cancellation occurs $x$ close to $\pi/2$
- use additional table for the final 512ulp before $\pi/2$

- denominator approximated by:

$$d = 1 - (\tan(a) + b)\tan(c)$$

# Implementation using Hard FP DSP Blocks

- Alignment:
  - $c$ and $a$ are obtained from $x_{fxp}$
  - reduced barrel shifter size: use top 18 bits of the $M_x$
  - maximum shift size is 13 ($tan(x) = x$ if required shift $> 13$)

# Implementation using Hard FP DSP Blocks

- Alignment:
  - $c$ and $a$ are obtained from $x_{fxp}$
  - reduced barrel shifter size: use top 18 bits of the $M_x$
  - maximum shift size is 13 ($tan(x) = x$ if required shift $> 13$)
- B in floating-point:
  - use mask to extract FP information

| $e$ | $e_{biased}$ | mask |
|---|---|---|
| 0 | 01111111 | 00000000000000000111111 |
| -1 | 01111110 | 00000000000000001111111 |
| -2 | 01111101 | 00000000000000011111111 |
| -3 | 01111100 | 00000000000000111111111 |
| -4 | 01111011 | 00000000000001111111111 |
| -5 | 01111010 | 00000000000011111111111 |
| -6 | 01111001 | 00000000000111111111111 |
| -7 | 01111000 | 00000000001111111111111 |
| -8 | 01110111 | 00000000011111111111111 |
| -9 | 01110110 | 00000000111111111111111 |
| -10 | 01110101 | 00000001111111111111111 |
| -11 | 01110100 | 00000011111111111111111 |
| -12 | 01110011 | 00000111111111111111111 |
| -13 | 01110010 | 00001111111111111111111 |
| ... | ... | ... |

  - use FP subtraction to get B in FP.

# Implementation using Hard FP DSP Blocks

- Alignment:
  - $c$ and $a$ are obtained from $x_{fxp}$
  - reduced barrel shifter size: use top 18 bits of the $M_x$
  - maximum shift size is 13 ($tan(x) = x$ if required shift $> 13$)
- B in floating-point:
  - use mask to extract FP information

| $e$ | $e_{biased}$ | mask |
|---|---|---|
| 0 | 01111111 | 00000000000000000 111111 |
| -1 | 01111110 | 00000000000000001 111111 |
| -2 | 01111101 | 00000000000000011 111111 |
| -3 | 01111100 | 00000000000000111 111111 |
| -4 | 01111011 | 00000000000001111 111111 |
| -5 | 01111010 | 00000000000011111 111111 |
| -6 | 01111001 | 00000000000111111 111111 |
| -7 | 01111000 | 00000000001111111 111111 |
| -8 | 01110111 | 00000000011111111 111111 |
| -9 | 01110110 | 00000000111111111 111111 |
| -10 | 01110101 | 00000001111111111 111111 |
| -11 | 01110100 | 00000011111111111 111111 |
| -12 | 01110011 | 00000111111111111 111111 |
| -13 | 01110010 | 00000111111111111 111111 |
| ... | ... | ... |

  - use FP subtraction to get B in FP.

# Implementation using Hard FP DSP Blocks

- Alignment:
  - $c$ and $a$ are obtained from $x_{fxp}$
  - reduced barrel shifter size: use top 18 bits of the $M_x$
  - maximum shift size is 13 ($tan(x) = x$ if required shift $> 13$)
- B in floating-point:
  - use mask to extract FP information

| $e$ | $e_{biased}$ | mask |
|---|---|---|
| 0 | 0111 1111 | 00000000000000000 111111 |
| -1 | 0111 1110 | 00000000000000001 111111 |
| -2 | 0111 1101 | 00000000000000011 111111 |
| -3 | 0111 1100 | 00000000000000111 111111 |
| -4 | 0111 1011 | 00000000000001111 111111 |
| -5 | 0111 1010 | 00000000000011111 111111 |
| -6 | 0111 1001 | 00000000000111111 111111 |
| -7 | 0111 1000 | 00000000001111111 111111 |
| -8 | 0111 0111 | 00000000011111111 111111 |
| -9 | 0111 0110 | 00000000111111111 111111 |
| -10 | 0111 0101 | 00000001111111111 111111 |
| -11 | 0111 0100 | 00000011111111111 111111 |
| -12 | 0111 0011 | 00000111111111111 111111 |
| -13 | 0111 0010 | 00001111111111111 111111 |
| ... | ... | ... |

  - use FP subtraction to get B in FP.

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 00000000001111111

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 00000000001111111

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 00000000001111111 & 111111 right padding

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 00000000001111111 & 111111 right padding
- apply mask on fraction:
  10011110001101001111001 AND
  00000000001111111111111 =
  ---
  00000000001101001111001

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 00000000001111111 & 111111 right padding
- apply mask on fraction:
  10011110001101001111001 AND
  00000000001111111111111 =
  ---
  00000000001101001111001
- create FP value $v_l$=0 01111000 00000000001101001111001
- create FP value $v_r$=0 01111000 00000000000000000000000
- obtain $b = v_l - v_r = 2^{-18}1.10100111100100000000000_2$
- $b$ = 0 01101101 10100111100100000000000

# Obtaining $b$ in FP example

- let $x$ = 0 01111000 10011110001101001111001
- mask is 0000000001111111 & 111111 right padding
- apply mask on fraction:
  10011110001101001111001 AND
  00000000001111111111111 =
  ---
  00000000001101001111001
- create FP value $v_l$=0 01111000 00000000001101001111001
- create FP value $v_r$=0 01111000 00000000000000000000000
- obtain $b = v_l - v_r = 2^{-18}1.10100111100100000000000_2$
- $b$ = 0 01101101 10100111100100000000000
- this is the same as:

| c (9 bits) | a (9 bits) | b (18 bits) |
|---|---|---|
| 000000011 | 001111000 | 11010011110010000 |

# Implementation using Hard FP DSP Blocks

- Common-subexpression
  - $\tan(a) + b$ both in numerator and denominator
  - obtain $\tan(a)$ by tabulation directly in FP
  - perform sum in SP FP

# Implementation using Hard FP DSP Blocks

- Common-subexpression
  - $\tan(a) + b$ both in numerator and denominator
  - obtain $\tan(a)$ by tabulation directly in FP
  - perform sum in SP FP
- Numerator
  - obtain $\tan(c)$ by tabulation directly in FP
  - use a FP adder to add $\tan(c)$ to the value of the common-subexpression

# Implementation using Hard FP DSP Blocks

- Common-subexpression
  - $\tan(a) + b$ both in numerator and denominator
  - obtain $\tan(a)$ by tabulation directly in FP
  - perform sum in SP FP
- Numerator
  - obtain $\tan(c)$ by tabulation directly in FP
  - use a FP adder to add $\tan(c)$ to the value of the common-subexpression
- Denominator
  - multiply $\tan(c)$ by value of common-subexpression
  - use a FP subtracter to perform $1 - (\tan(a) + b)\tan(c)$

# Implementation using Hard FP DSP Blocks

- Common-subexpression
  - $\tan(a)+b$ both in numerator and denominator
  - obtain $\tan(a)$ by tabulation directly in FP
  - perform sum in SP FP
- Numerator
  - obtain $\tan(c)$ by tabulation directly in FP
  - use a FP adder to add $\tan(c)$ to the value of the common-subexpression
- Denominator
  - multiply $\tan(c)$ by value of common-subexpression
  - use a FP subtracter to perform $1-(\tan(a)+b)\tan(c)$
- Reciprocal
  - can use a piecewise-polynomial-based inverse

# Implementation using Hard FP DSP Blocks

- Common-subexpression
  - $\tan(a)+b$ both in numerator and denominator
  - obtain $\tan(a)$ by tabulation directly in FP
  - perform sum in SP FP
- Numerator
  - obtain $\tan(c)$ by tabulation directly in FP
  - use a FP adder to add $\tan(c)$ to the value of the common-subexpression
- Denominator
  - multiply $\tan(c)$ by value of common-subexpression
  - use a FP subtracter to perform $1-(\tan(a)+b)\tan(c)$
- Reciprocal
  - can use a piecewise-polynomial-based inverse
- Assembly
  - use a floating-point multiplier

# Enhanced reciprocal exponent handling

- compute reciprocal of denominator:
  - mantissa $M_d = 1 + y$, $y \in [0, 1)$
  - the exponent of the denominator $e_d^{biased} = e_d + bias$.
- the reciprocal:
  - mantissa $1/M_d \in (0.5, 1]$
  - the pre-normalization biased reciprocal exponent:

$$\begin{aligned} e_r^{biased} &= -e_d + bias \\ &= 2bias - e_d^{biased} \end{aligned}$$

# Enhanced reciprocal exponent handling

- compute reciprocal of denominator:
  - mantissa $M_d = 1 + y$, $y \in [0, 1)$
  - the exponent of the denominator $e_d^{biased} = e_d + bias$.
- the reciprocal:
  - mantissa $1/M_d \in (0.5, 1]$
  - the pre-normalization biased reciprocal exponent:

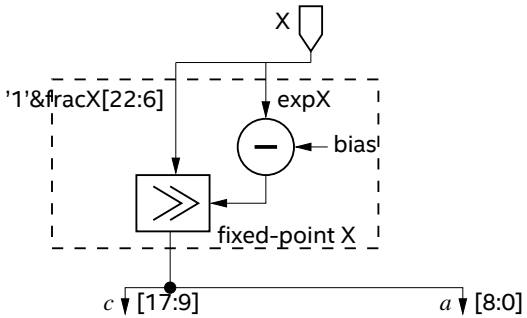
$$\begin{aligned} e_r^{biased} &= -e_d + bias \\ &= 2bias - e_d^{biased} \end{aligned}$$

- normalization $\rightarrow$ exponent adjustment
  - let $n$ be true if need to normalize, or $1/M_d \in (0.5, 1)$.
  - $e_{rPostNorm}^{biased} = 2bias - e_d^{biased} - n$.
  - use bit manipulations to directly get $2bias - n$

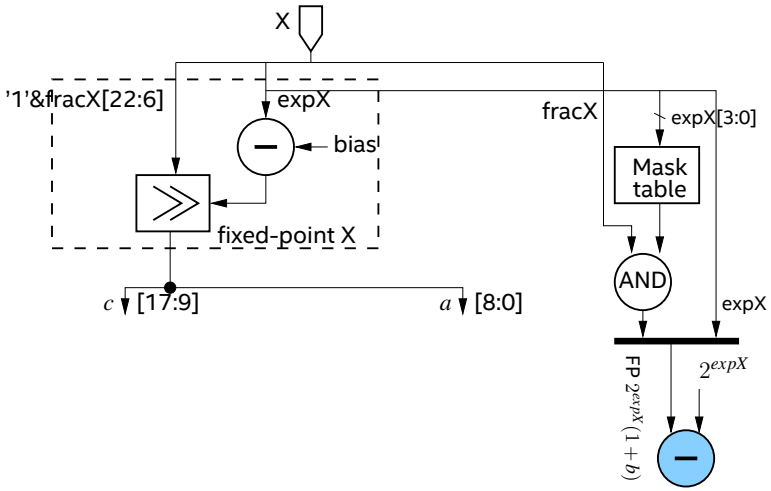
$$\begin{aligned} 2bias - n &= 11111110 - n \\ &= 111111\widetilde{n}n \end{aligned}$$

# Architecture

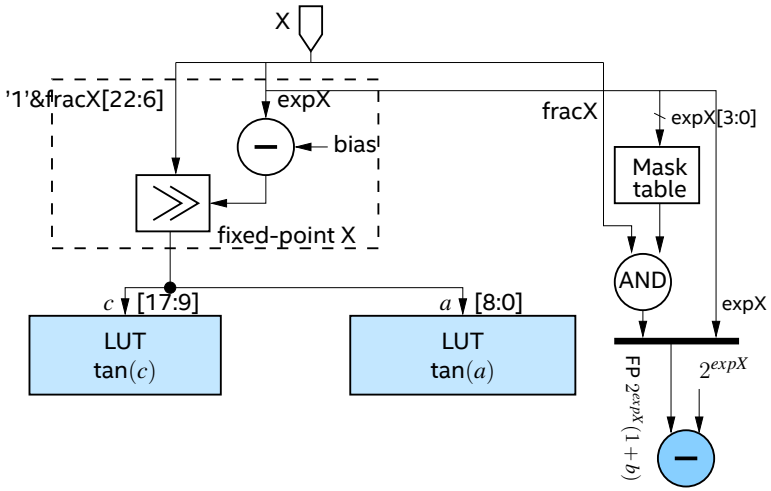
# Architecture



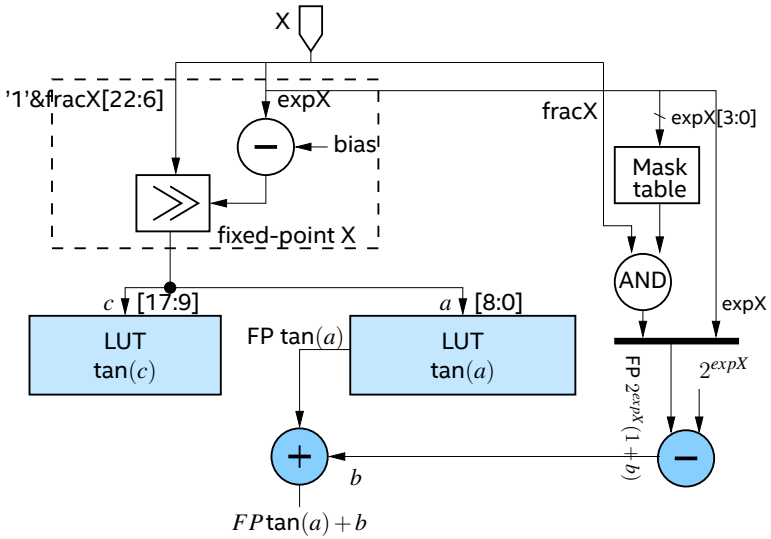
# Architecture



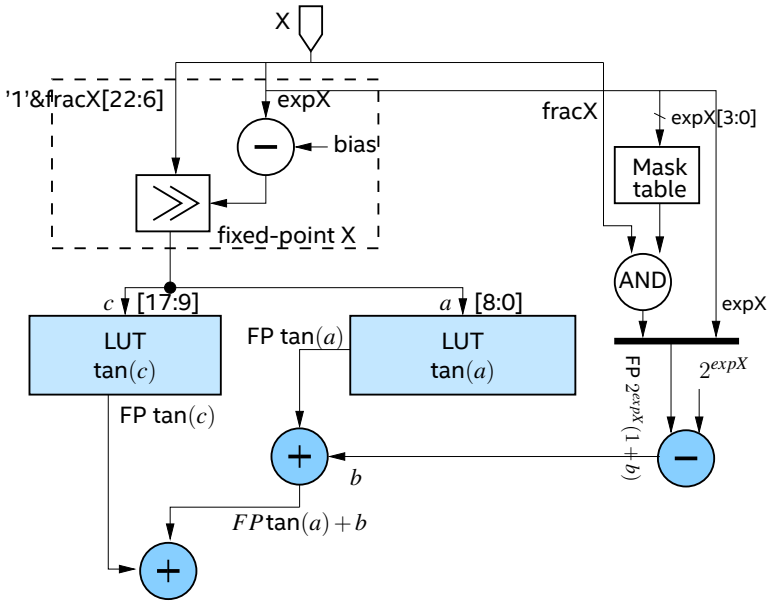
# Architecture



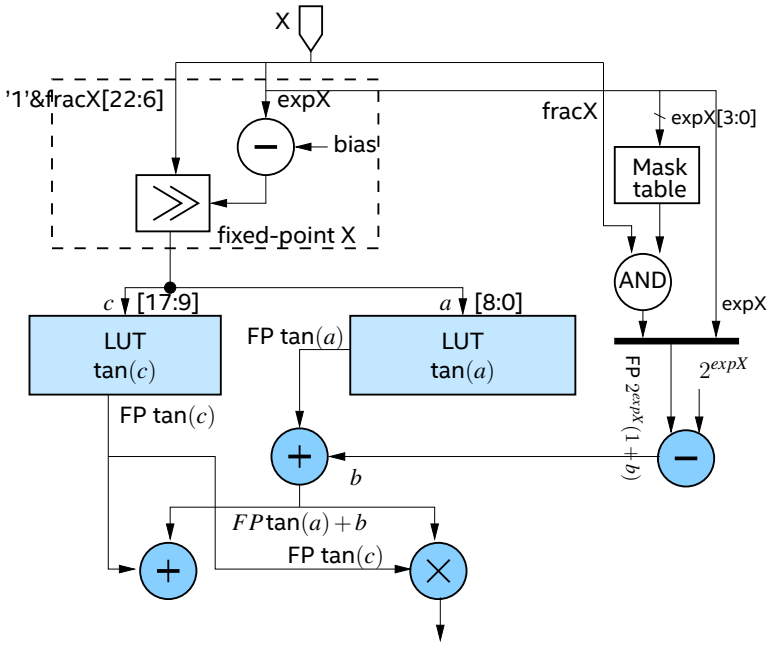
# Architecture



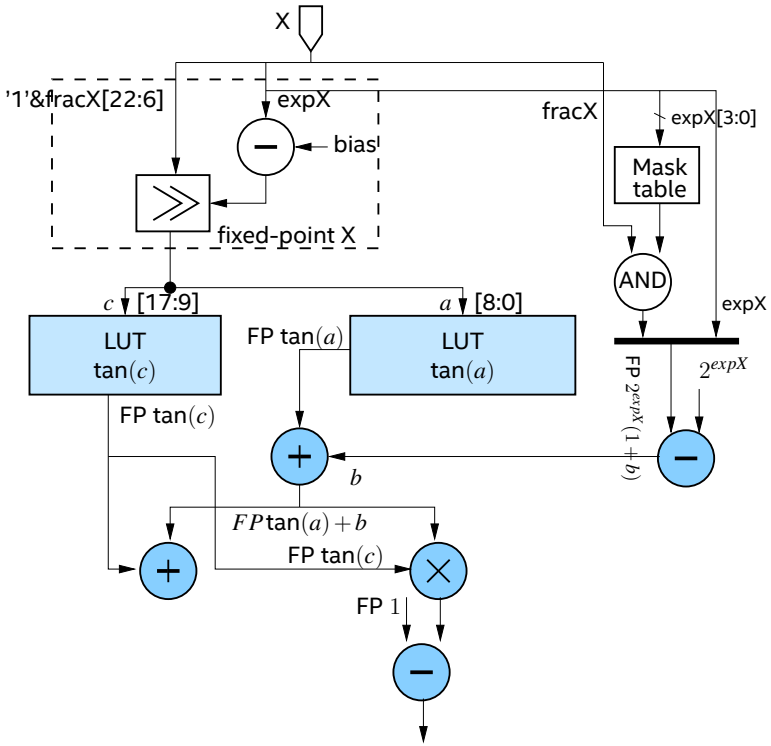
# Architecture



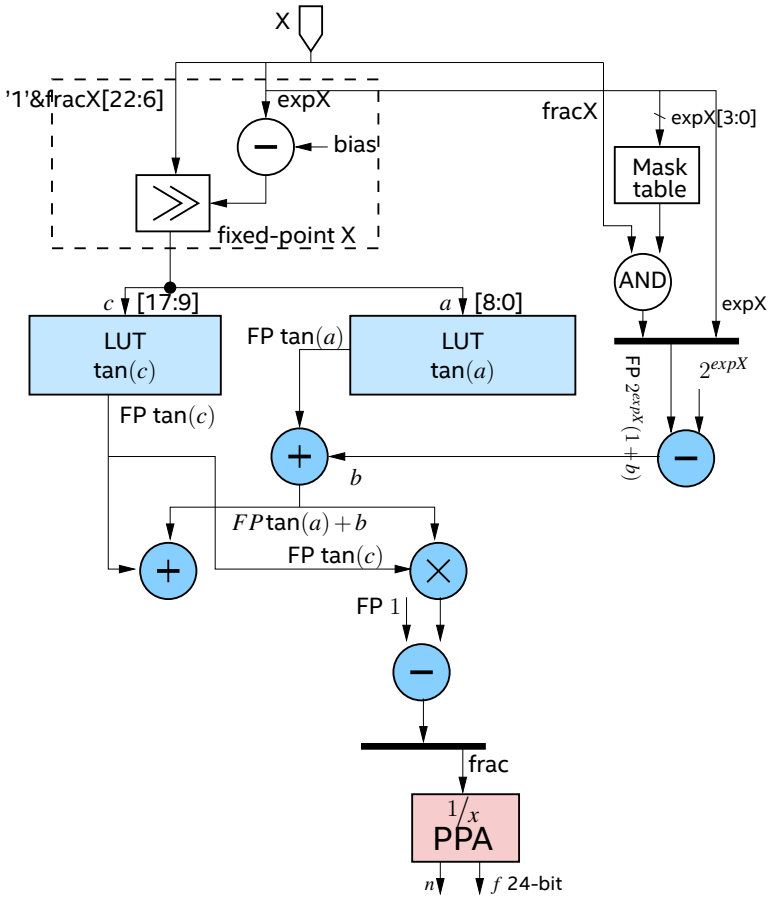
# Architecture



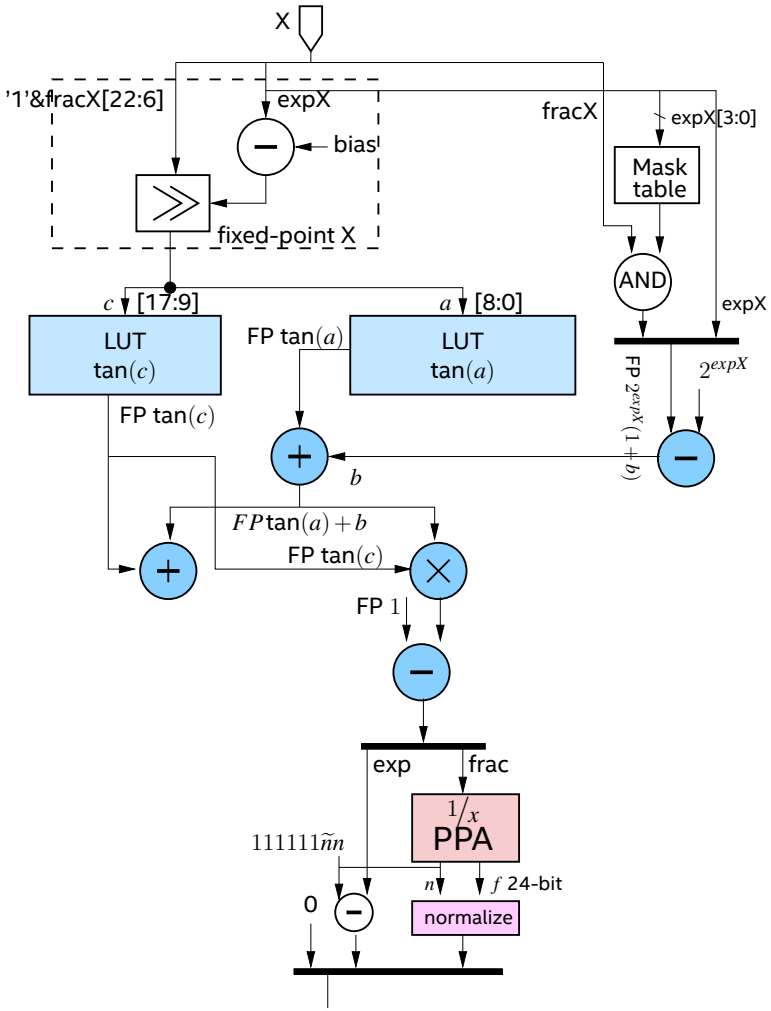
# Architecture



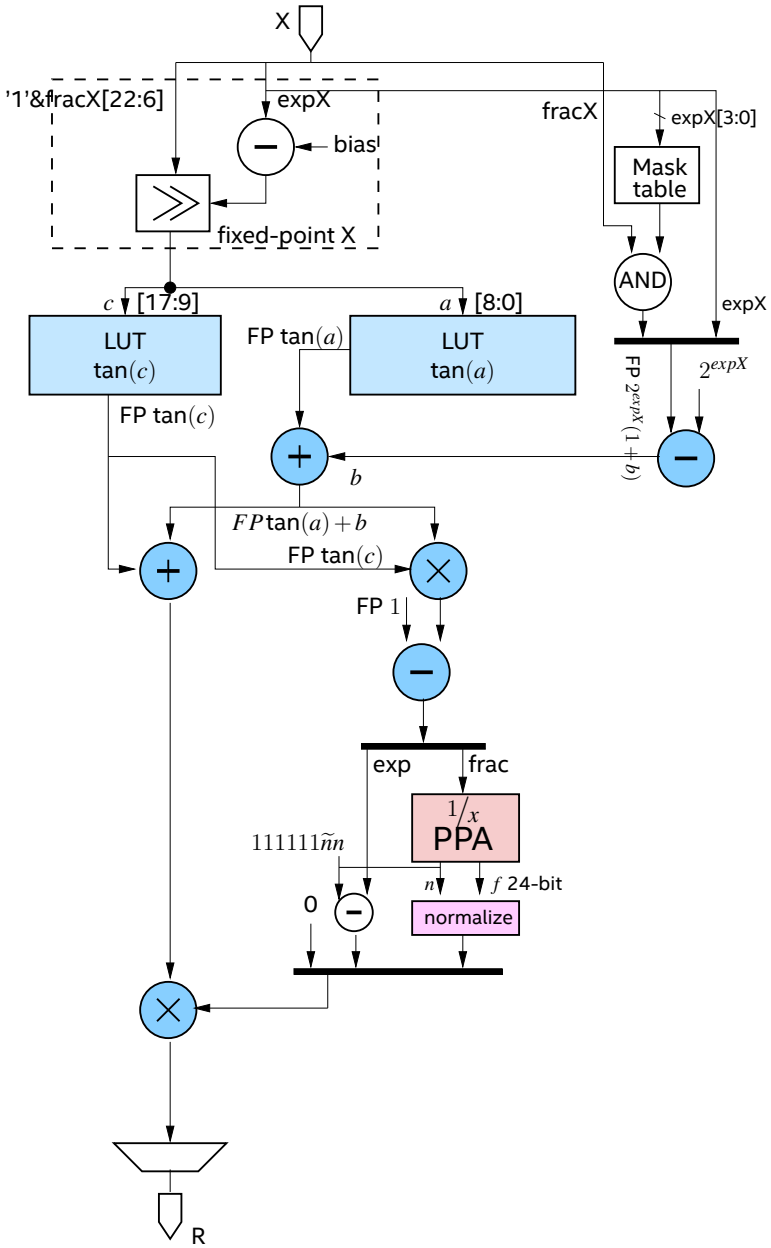
# Architecture



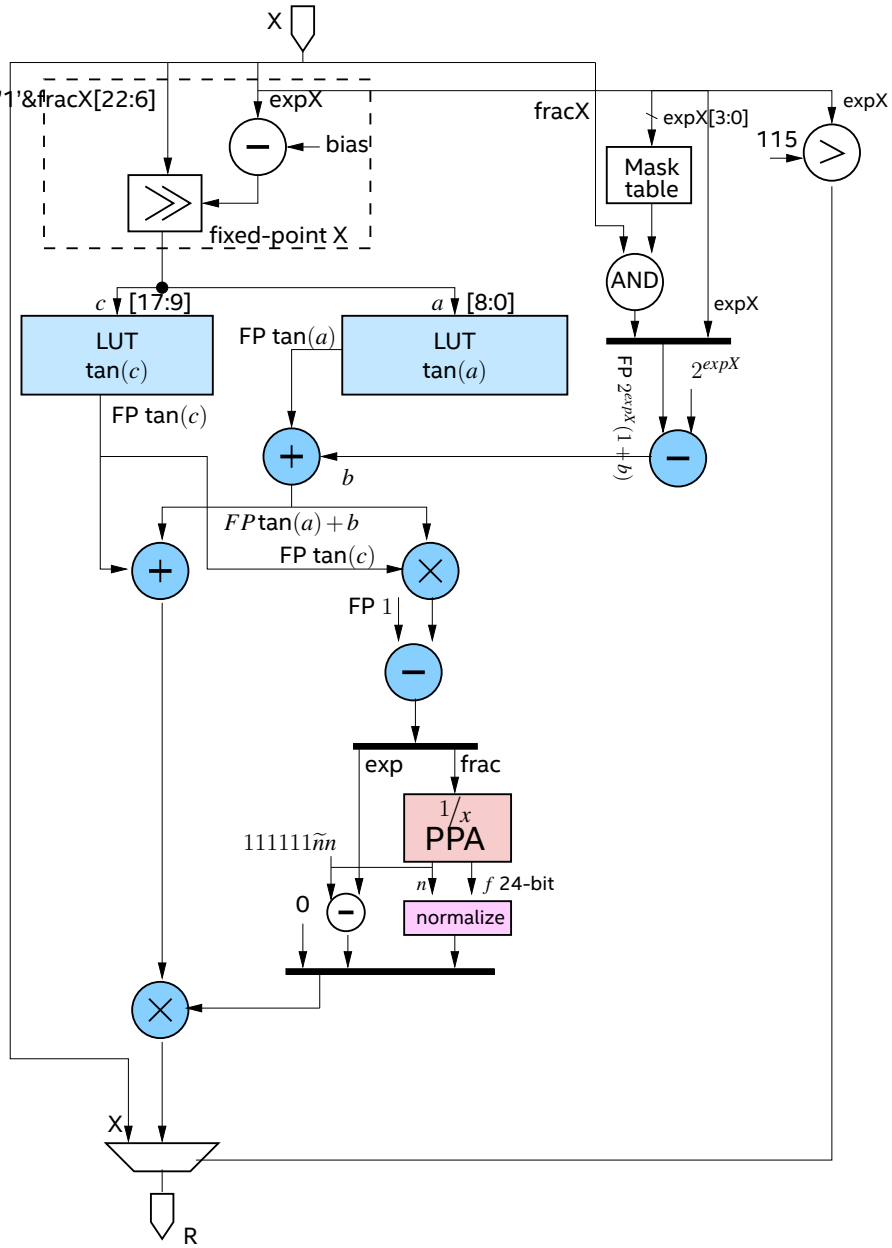
# Architecture



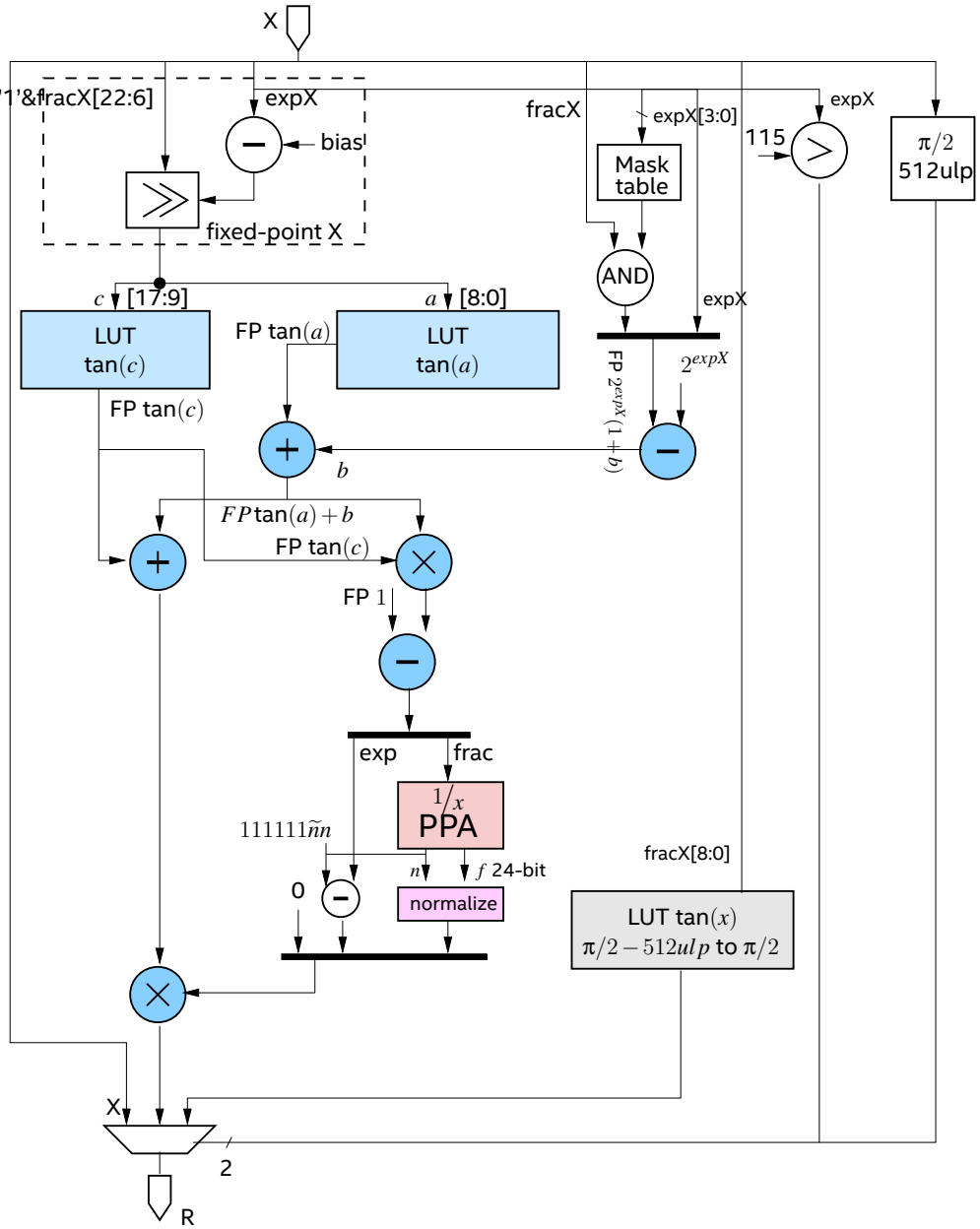
# Architecture



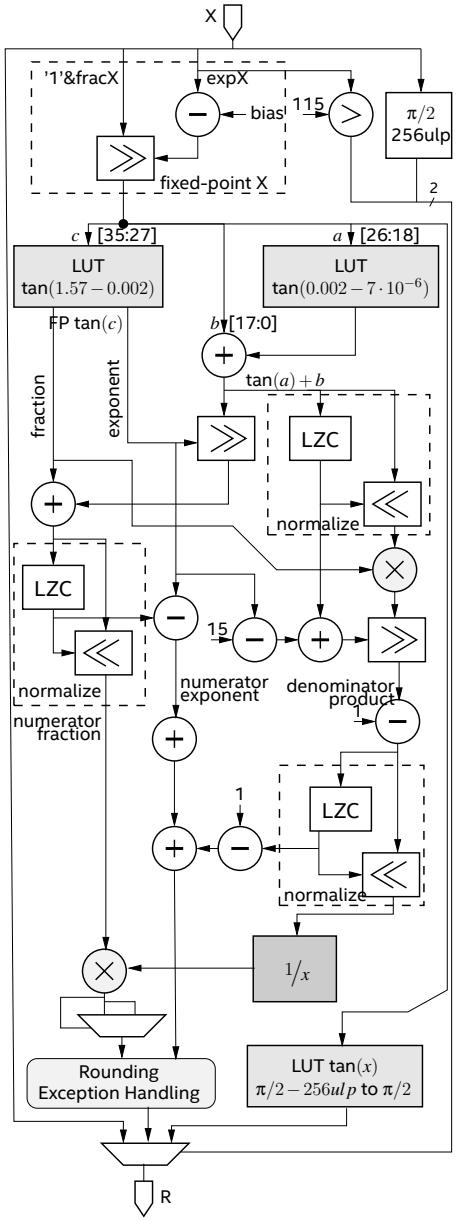
# Architecture

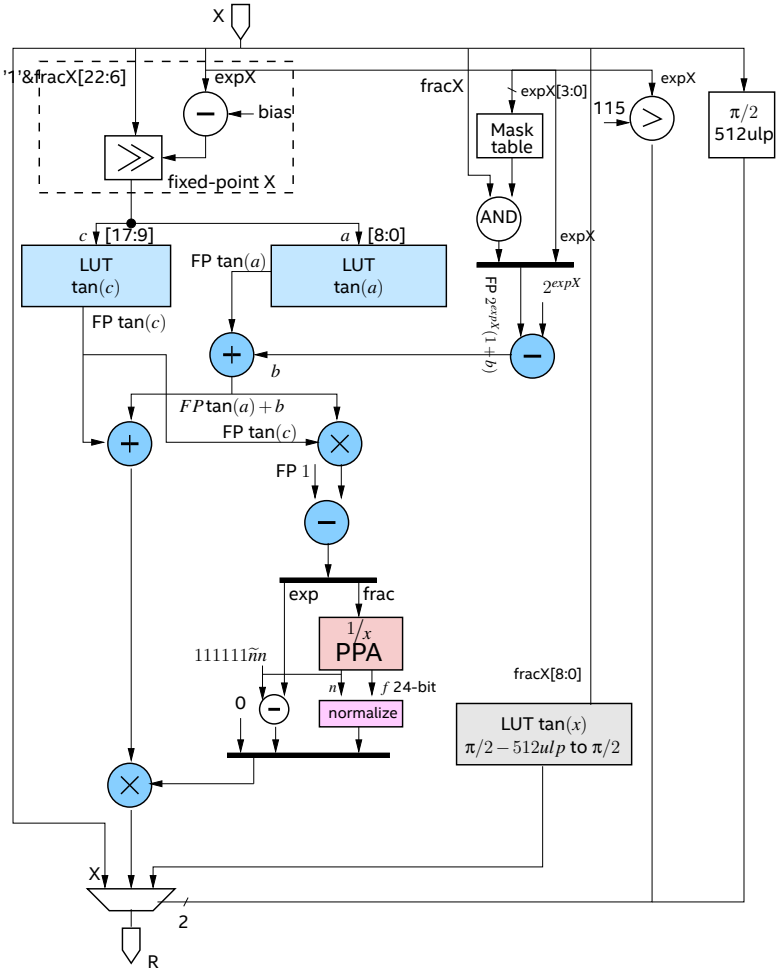


# Architecture



# Side-by-side comparison against non-HFP architecture



# Results

Table: Synthesis results for recent FPGA architectures

| Target | Architecture | Latency @ Freq. | Resources |
|---|---|---|---|
| Arria 10 | Proposed | 36 @ 520MHz | 8 DSPs, 6 M20K, 990ALMs |
| Stratix 10 | Proposed w/o HIPI | 53 @ 532MHz | 8 DSPs, 7 M20K, 1573ALMs |
| Stratix 10 | Proposed HIPI * | 53 @ 655MHz | 8 DSPs, 7 M20K, 1632ALMs |
| Arria 10 | Proposed FP | 32 @ 487MHz | 8 DSPs, 6 M20K, 393ALMs |
| Stratix 10 | Proposed FP w/o HIPI | 39 @ 573MHz | 8 DSPs, 6 M20K, 566ALMs |
| Stratix 10 | Proposed FP HIPI * | 39 @ 644MHz | 8 DSPs, 6 M20K, 572ALMs |

- results for both proposed architectures
- throughput of 1 operation / clock cycle
- largest Arria 10 472200 ALMs
  - Utilization of 0.23% for Proposed, 0.09% Proposed FP
- same Arria 10 1518 DSPs
  - Utilization of 0.5% for Proposed and Proposed FP
- same Arria 10 2713 M20Ks
  - Utilization of 0.2% for Proposed and Proposed FP
- HyperFlex architecture enhances Stratix 10 performance

# Results

Table: Synthesis results for recent FPGA architectures

| Target | Architecture | Latency @ Freq. | Resources |
|---|---|---|---|
| Arria 10 | Proposed | 36 @ 520MHz | 8 DSPs, 6 M20K, 990ALMs |
| Stratix 10 | Proposed w/o HIPI | 53 @ 532MHz | 8 DSPs, 7 M20K, 1573ALMs |
| Stratix 10 | Proposed HIPI * | 53 @ 655MHz | 8 DSPs, 7 M20K, 1632ALMs |
| Arria 10 | Proposed FP | 32 @ 487MHz | 8 DSPs, 6 M20K, 393ALMs |
| Stratix 10 | Proposed FP w/o HIPI | 39 @ 573MHz | 8 DSPs, 6 M20K, 566ALMs |
| Stratix 10 | Proposed FP HIPI * | 39 @ 644MHz | 8 DSPs, 6 M20K, 572ALMs |

- results for both proposed architectures
- throughput of 1 operation / clock cycle
- largest Arria 10 472200 ALMs
  - Utilization of 0.23% for Proposed, 0.09% Proposed FP
- same Arria 10 1518 DSPs
  - Utilization of 0.5% for Proposed and Proposed FP
- same Arria 10 2713 M20Ks
  - Utilization of 0.2% for Proposed and Proposed FP
- HyperFlex architecture enhances Stratix 10 performance
- Stratix 10 ALMs $\times$ 2, DSPs $\times$ 3.8, M20Ks $\times$ 4.3

# Conclusion

- proposed fused-operator for single-precision tangent
- error analysis allows meeting OpenCL accuracy requirements
- proposed cores are fully pipelined, throughput 1 operation/cycle
- proposed cores require very little area
- DSP and Memory centric architectures allow reaching high frequencies
- Stratix 10 and Arria 10 Hard FP DSP block allows for an efficient implementation
- lower latency in high-frequency scenarios

# References


[1] de Dinechin, F., and Pasca, B.
Designing custom arithmetic data paths with FloPoCo.
IEEE Design and Test (2011).

[2] Detrey, J., and de Dinechin, F.
Floating-point trigonometric functions for FPGAs.
In International Conference on Field Programmable Logic and Applications (Amsterdam, Netherlands, aug 2007), IEEE, pp. 29–34.

[3] Garcia, E., Cumplido, R., and Arias, M.
Pipelined CORDIC design on FPGA for a digital sine and cosine waves generator.
In Electrical and Electronics Engineering, 2006 3rd International Conference on (sept. 2006), pp. 1 –4.

[4] Langhammer, M., and Pasca, B.
Efficient floating-point polynomial evaluation on FPGAs.
In 22th International Conference on Field Programmable Logic and Applications (FPL'13) (Porto, Portugal, Aug. 2013), IEEE.

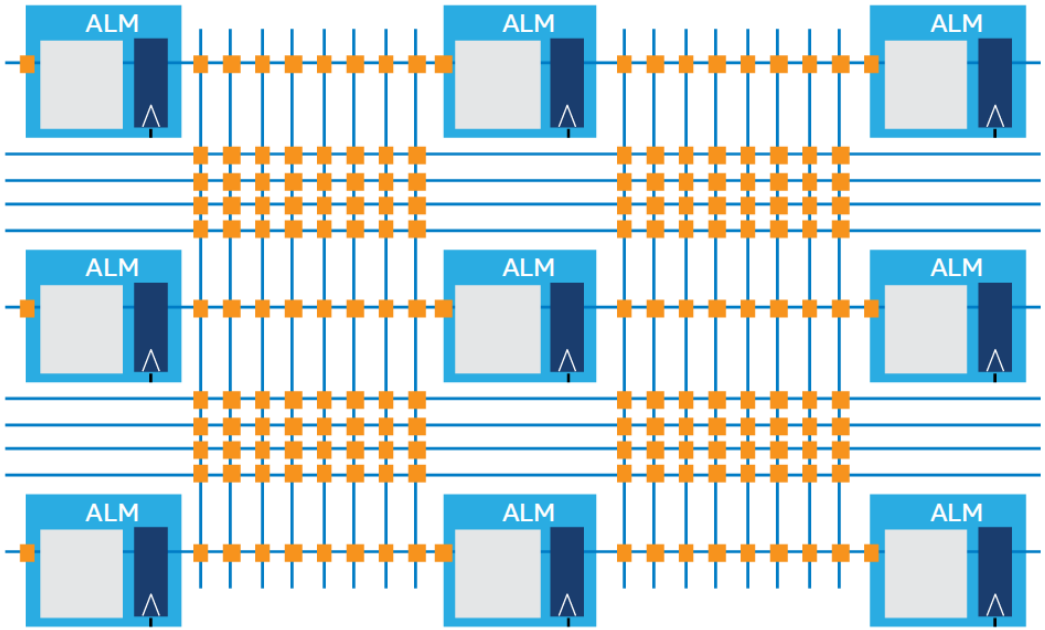
[5] Langhammer, M., and VanCourt, T.
FPGA floating point datapath compiler.
Field-Programmable Custom Computing Machines, Annual IEEE Symposium on 17 (2009), 259–262.

[6] Pasca, B.
Correctly rounded floating-point division for DSP-enabled FPGAs.
In 22th International Conference on Field Programmable Logic and Applications (FPL'12) (Oslo, Norway, Aug. 2012), IEEE.

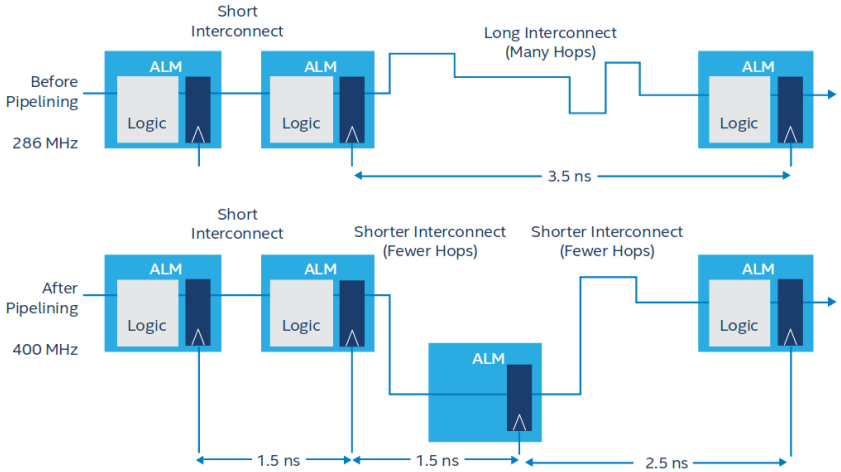
[7] Payne, M. H., and Hanek, R. N.
Radian reduction for trigonometric functions.
ACM SIGNUM Newsletter 18, 1 (Jan. 1983), 19–24.

[8] Shang, Y.
Implementation of ip core of fast sine and cosine operation through FPGA.
Energy Procedia 16, Part B, 0 (2012), 1253 – 1258.
2012 ICFEEM.

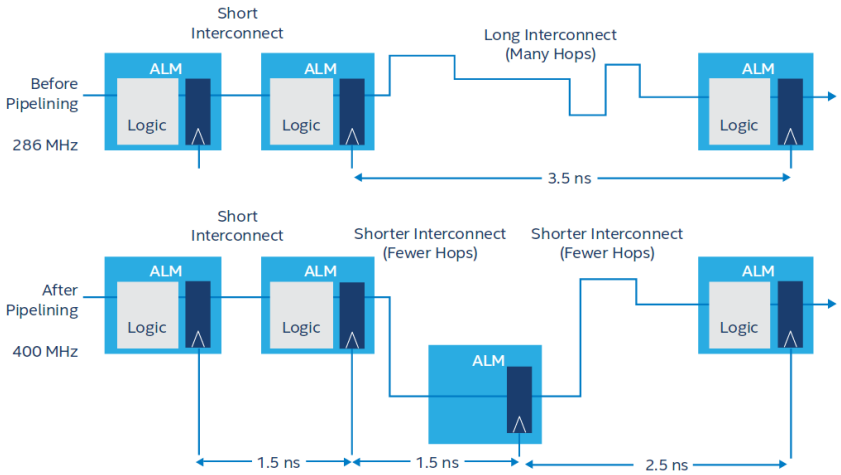
Figure: ”Registers Everywhere” HyperFlex Architecture



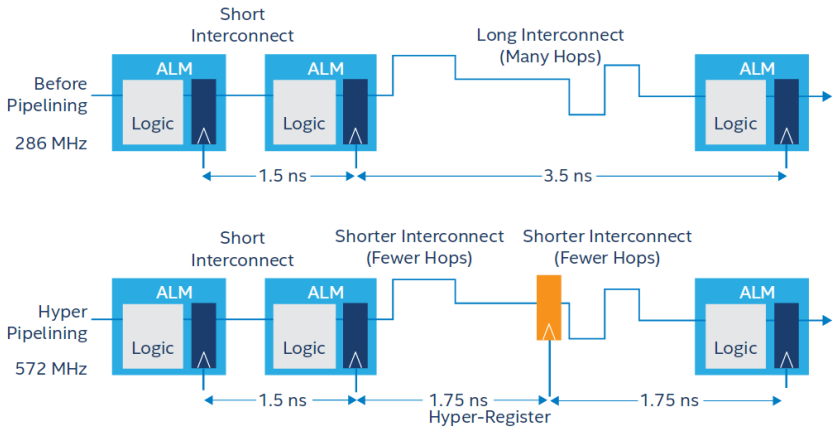
# Figure: Pipelining in Conventional FPGA Architectures



## Figure: Pipelining in Conventional FPGA Architectures




## Figure: Hyper-Pipelining in the HyperFlex Architecture




back