# A High Throughput Polynomial and Rational Function Approximations Evaluator

**Silviu-Ioan Filip**, Univ Rennes, Inria, CNRS, IRISA

Joint work with **N. Brisebarre, G. Constantinides, M. Ercegovac, M. Iștoan & J-M. Muller**

**25th IEEE Symposium on Computer Arithmetic**
**Amherst, MA, June 2018**

# Context

**Mathematical function evaluation**

## How?

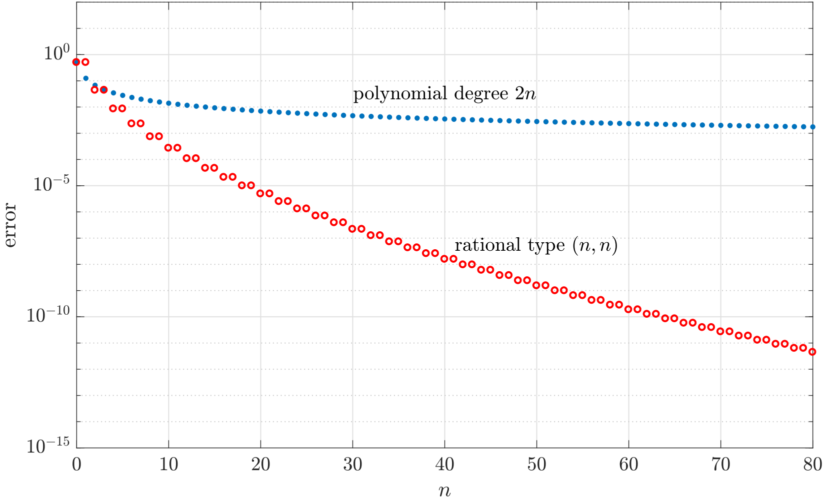
- in particular cases, use *ad hoc* solutions
  - e.g. CORDIC, tabulate-and-compute algorithms
- in general, polynomial and rational function approximations

Rational functions are more powerful than polynomials

# Polynomials vs rational approximations

Celebrated example in approximation theory: $f(x) = |x|, x \in [-1, 1]$

- polynomial $2n$: $O(1/n)$ [Bernstein 1910s, Varga & Carpenter 1985]
- rational $(n, n)$: $O(\exp(-\pi\sqrt{n}))$ [Newmann 1964, Stahl 1993]



# Context

**Mathematical function evaluation**

### How?

- in particular cases, use *ad hoc* solutions
  - e.g. CORDIC, tabulate-and-compute algorithms
- in general, polynomial and rational function approximations

Rational functions are more powerful than polynomials

# Context

**Mathematical function evaluation**

**How?**

- in particular cases, use *ad hoc* solutions
  - e.g. CORDIC, tabulate-and-compute algorithms
- in general, polynomial and rational function approximations

Rational functions are more powerful than polynomials

**Our goal:**

**Benefits to implementing rational functions in HW?**

# The E-method [Ercegovac 1975, 1977]

$$R(x) = \frac{p_\mu x^\mu + p_{\mu-1} x^{\mu-1} + \cdots + p_0}{q_\nu x^\nu + q_{\nu-1} x^{\nu-1} + \cdots + 1}$$

**Idea**:

- $R(x)$ mapped to a linear system: $\mathbf{A}_x \mathbf{y} = \mathbf{p}$

$$\begin{bmatrix} 1 & -x & 0 & & \cdots & 0 \\ q_1 & 1 & -x & 0 & \cdots & 0 \\ q_2 & 0 & 1 & -x & \cdots & 0 \\ & & \ddots & \ddots & & \vdots \\ \vdots & & & & \ddots & 0 \\ q_{n-1} & & & & 1 & -x \\ q_n & & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_{n-1} \\ p_n \end{bmatrix}$$

$$\boxed{y_0 = R(x)}$$

$$n = \max\{\mu, \nu\}, p_k = 0, \mu < k \leqslant n, q_k = 0, \nu < k \leqslant n$$

# The E-method [Ercegovac 1975, 1977]

- digit-by-digit computation

$$\mathbf{w}^{(j)} = r \cdot \left[\mathbf{w}^{(j-1)} - \mathbf{A}_x \mathbf{d}^{(j-1)}\right], \qquad j = 1, \ldots, m$$

$m$ - number of iterations
$\mathbf{d}^{(0)} = \mathbf{0}, \mathbf{w}^{(0)} = \mathbf{p}$
$d_i^{(j)}$ - single digit rounding of $w_i^{(j)}$

**Result:** $m$-digit solution in radix $r$

$$y_k = \sum_{j=1}^{m} d_k^{(j)} r^{-j}$$

**Rule of thumb:** *comparable* cost for deg $n$ poly & type $(n, n)$ rational

# Convergence of the E-method

$\rightarrow$ requires *bounds* on the $p_k$'s, $q_k$'s and approximation domain $[a, b]$

$$\begin{cases} \forall k, |p_k| \leq \xi, \\ \forall k, |x| + |q_k| \leq \alpha \end{cases}$$

$$\xi = \frac{1}{2}(1 + \mathit{\Delta}),$$

$$0 < \mathit{\Delta} < 1,$$

$$\alpha \leq (1 - \mathit{\Delta})/(2r)$$

- **polynomial**: always (up to scaling the $p_k$'s and change of variable)
- **rational**: *requires* bounded $|q_k|$'s $\rightarrow$ E-fraction approximations

# Computing E-fraction approximations

**Input:** $f \in \mathcal{C}([a,b]), \mu, \nu \in \mathbb{N}$, magnitude bound $d > 0$.

**Output:** $R(x) = \dfrac{p_\mu x^\mu + p_{\mu-1}x^{\mu-1} + \cdots + p_0}{q_\nu x^\nu + q_{\nu-1}x^{\nu-1} + \cdots + 1}$, with $\max_{1 \leqslant k \leqslant \nu} |q_k| \leqslant d$,

s.t.

$$\max_{x \in [a,b]} |f(x) - R(x)|$$

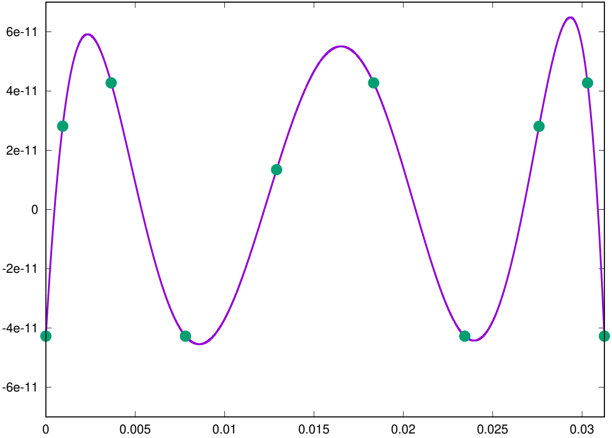
is **minimal**.

### How?

- $\nu = 0 \rightarrow$ use the polynomial Remez exchange algorithm [Remez 1934]
- $\nu > 0 \rightarrow$ we ***developed*** a greedy-based iterative algorithm

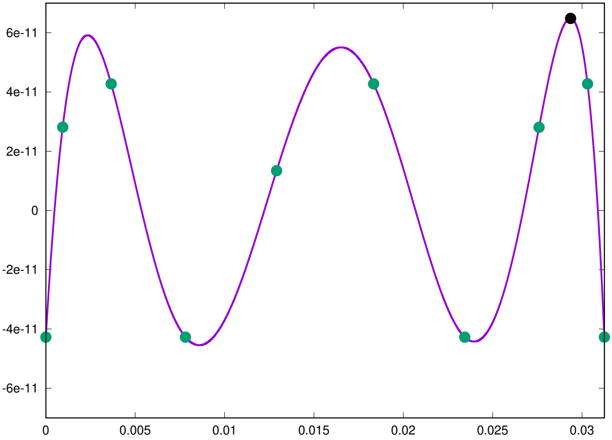
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 1



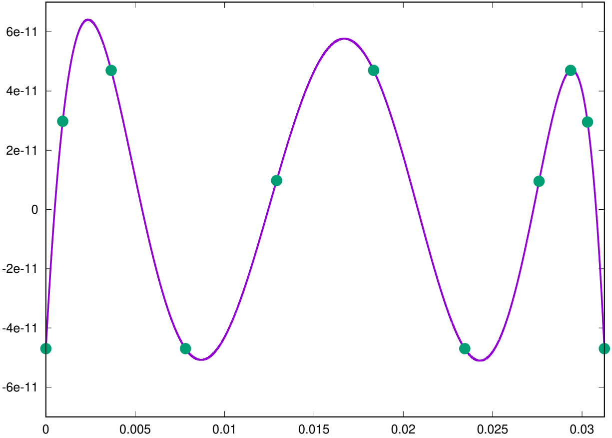
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** $1$



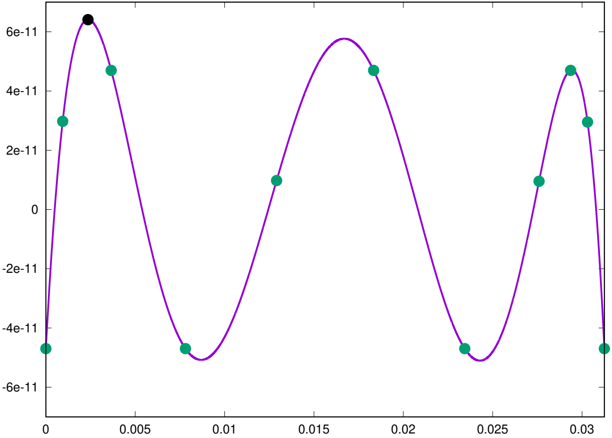
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 2



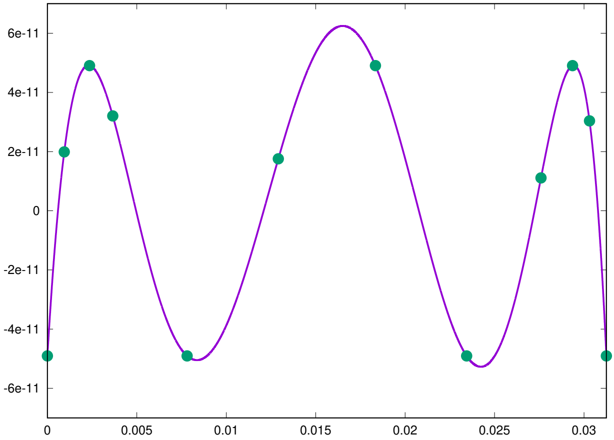
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\varDelta = 1/8$ and $r = 2$)
**Iteration** 2



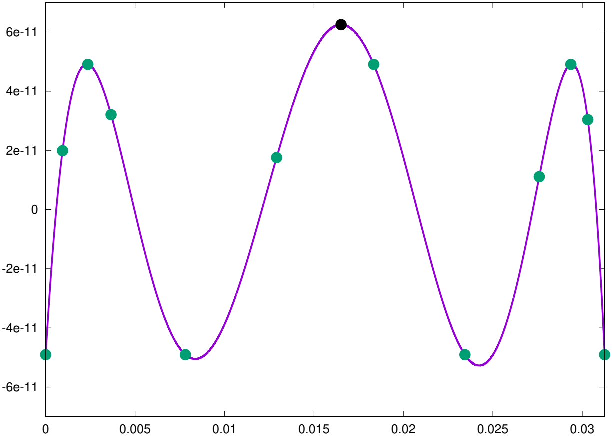
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 3



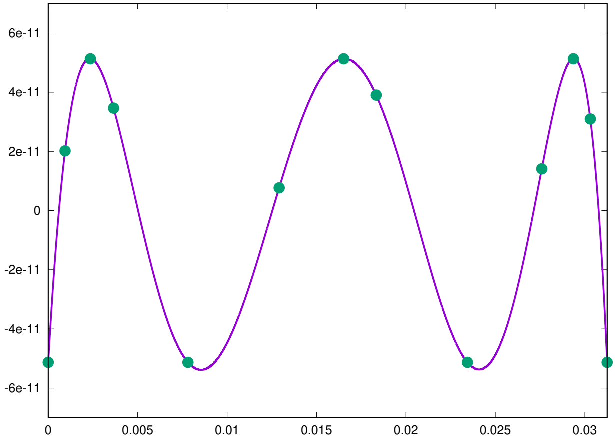
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 3



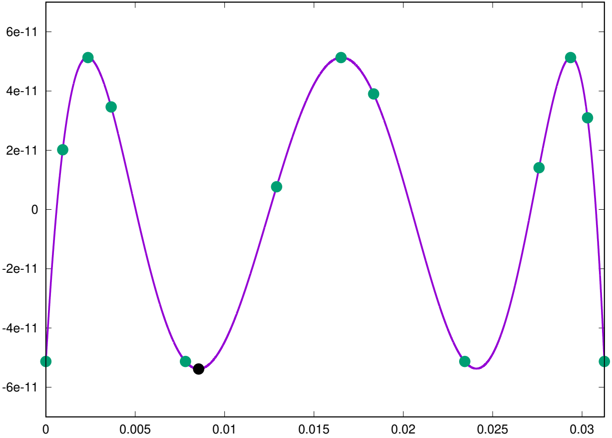
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 4



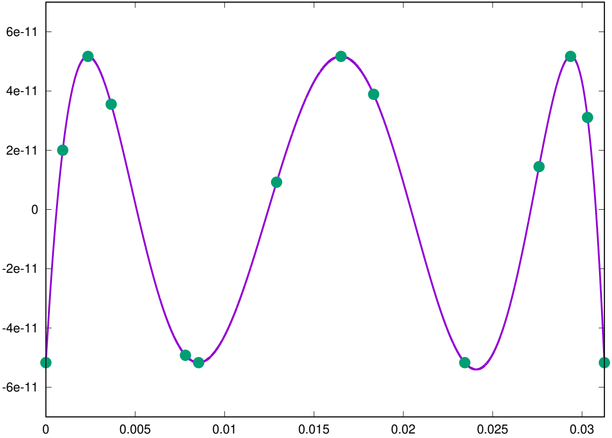
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 4



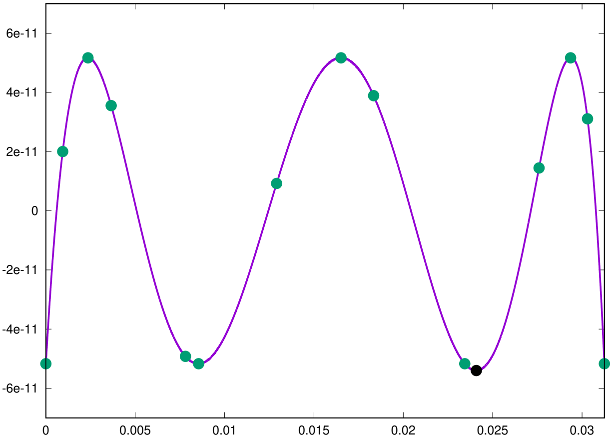
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 5



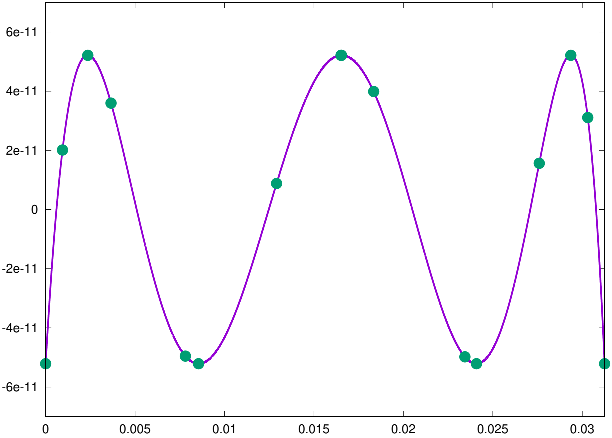
# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 5



# An example

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$, $(\mu, \nu) = (4, 4)$
$d = 3/16$ ($\Delta = 1/8$ and $r = 2$)
**Iteration** 6




**Approximation error** $\simeq \mathbf{5.21 \cdot 10^{-11}}$
Polynomial $(\mu, \nu) = (4, 0)$ error $\simeq \mathbf{3.41 \cdot 10^{-10}}$

# Machine coefficient E-fractions

**In practice**, finite precision

$\rightarrow$ we target fixed-point implementations

- target error $\leqslant 2^{-m}, m \in \mathbb{N}$
- coefficients of the form $i/2^m, -2^m \leqslant i \leqslant 2^m$

**Ex. 1:** $m = 32$, target error $2^{-m} \simeq 2.33 \cdot 10^{-10}$

- real-coefficient error $5.21 \cdot 10^{-11}$
- rounding error $\mathbf{1.11 \cdot 10^{-9}} > 2^{-m}$ ☹

# Machine coefficient E-fractions

**In practice**, finite precision

$\rightarrow$ we target fixed-point implementations

- target error $\leqslant 2^{-m}, m \in \mathbb{N}$
- coefficients of the form $i/2^m, -2^m \leqslant i \leqslant 2^m$

**Ex. 1:** $m = 32$, target error $2^{-m} \simeq 2.33 \cdot 10^{-10}$

- real-coefficient error $5.21 \cdot 10^{-11}$
- rounding error $\mathbf{1.11 \cdot 10^{-9}} > 2^{-m}$ ☹

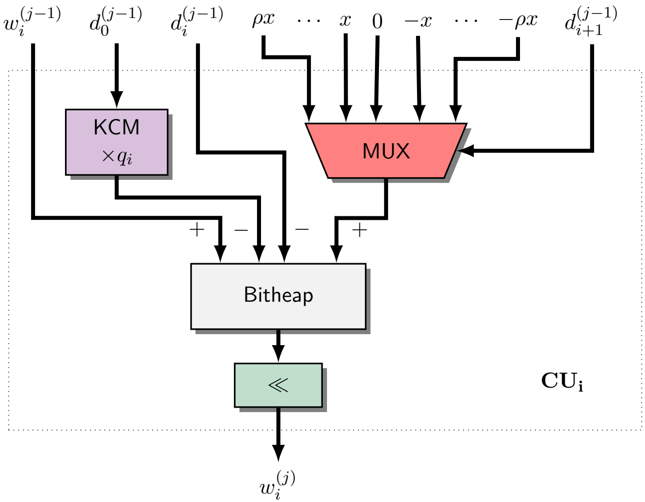
**Our approach:**

- based on [Brisebarre & Chevillard 2007, Brisebarre et al 2008]
- apply algorithms from Euclidean lattice theory
- lattice-based error $\simeq \mathbf{5.71 \cdot 10^{-11}} < 2^{-32}$ ☺

# HW implementation of the E-method

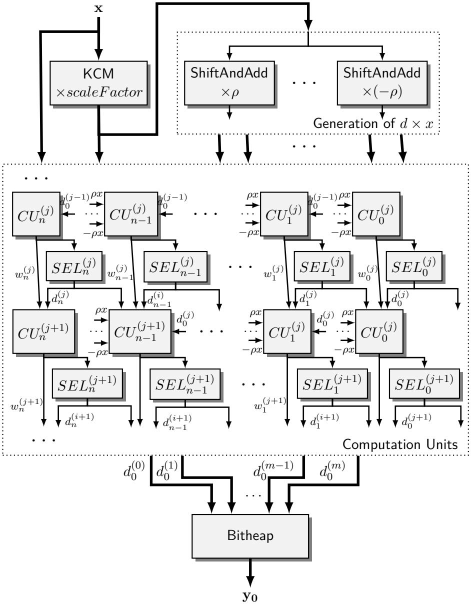
- generate circuit descriptions for FPGA devices
- flexibility in exploring different HW designs
- unrolled implementation of the method

# Architecture of an iteration




$$w_i^{(j)} = r \cdot \left[ w_i^{(j-1)} - q_i \cdot d_0^{(j-1)} - d_i^{(j-1)} + d_{i+1}^{(j-1)} \cdot x \right]$$

# Architecture of an unrolled implementation



# Optimizations

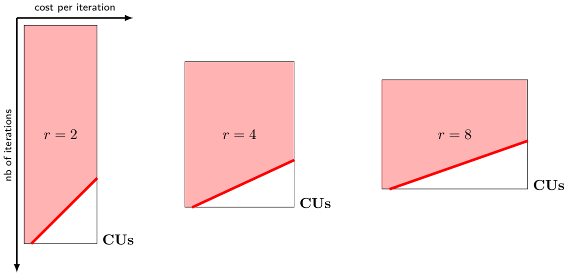
$$w_i^{(j)} = r \cdot \left[ w_i^{(j-1)} - q_i \cdot d_0^{(j-1)} - d_i^{(j-1)} + d_{i+1}^{(j-1)} \cdot x \right]$$

- $q_i \cdot d_0^{(j-1)}$
  - multiplication by a constant
  - use the *KCM technique* [Chapman 1993]
- $d_{i+1}^{(j-1)} \cdot x$
  - pre-compute all products
  - reduced to a MUX
- $r \cdot$
  - shift left by one digit
  - on an FPGA, no hardware needed (wiring)
- **summation**
  - use a *bitheap* [Brunie et al 2013]
    - generalization of a compressor tree
  - optimal cost and execution time

# Iteration-level optimizations

- **Iteration 0**:
  - initialization
  - store precomputed values
- **Iteration 1**:
  - precompute values
  - store in logic fabric
- **Iteration 2**:
  - precompute values (computations not involving $x$)
  - simpler iterations

## Simplifying the last iterations



# Some Results on a Xilinx Virtex6 device

**Ex. 1**: $f(x) = \sqrt{1 + (9x/2)^4}, x \in [0, 1/32]$

- target error $2^{-32}$, approx. type: $(4, 4)$

| Design | Approach | radix | Resources | | Performance |
|---|---|---|---|---|---|
| | | | LUT | reg. | cycles@period(ns) |
| Ex. 1 | Ours | 2 | 7,880 | 0 | 1@94.3 |
| | | | 7,966 | 1,523 | 11@9.6 |
| | | | 7,299 | 2,689 | 17@5.7 |
| | | | 6,786 | 5,202 | 36@3.7 |
| | | 4 | 4,871 | 0 | 1@57.9 |
| | | | 4,768 | 988 | 7@12.3 |
| | | | 4,600 | 1,583 | 11@6.9 |
| | | | 4,853 | 3,106 | 22@3.8 |
| | | 8 | 4,210 | 0 | 1@44.4 |
| | | | 3,875* | 0 | 1@62.2* |
| | | | 5,307* | 309 | 5@18.4* |
| | | | 5,184* | 499 | 8@10.4* |
| | | | 4,707* | 1,027 | 15@5.8* |
| | FloPoCo | – | 994 | 0 | 1@29.5 |
| | | | 1,032 | 138 | 7@6.7 |
| | | | 1,147 | 335 | 19@5.3 |

# More results

**Ex. 2**: $f(x) = \exp(2x)$, $x \in [0, 7/128]$

- target error $2^{-32}$, approx. types: $(3,3), (4,4), (5,0)$.

| Design | Approach | radix | Resources | | Performance |
|---|---|---|---|---|---|
| | | | LUT | reg. | cycles@period(ns) |
| Ex. 2 | Ours | 2 | 6,820 | 0 | 1@88.5 |
| | | 4 | 6,356 | 0 | 1@68.0 |
| | | 8 | 5,042 | 0 | 1@39.0 |
| | FloPoCo | – | 3,024 | 0 | 1@41.1 |

# Even more results

**Ex. 3**: $f(x) = \log_2\left(1 + 2^{-16x}\right), x \in [0, 1/16]$, approx. types: $(5,5), (5,0)$.
**Ex. 4**: $f(x) = \mathrm{erf}(x), x \in [0, 1/32]$, approx. types: $(4,4), (5,0)$.
**Ex. 5**: $f(x) = J_0(2x - 1/16), x \in [0, 1/16]$, approx. types: $(4,4), (6,0)$.

| Design | Approach | radix | Resources | | Performance | Target error |
|---|---|---|---|---|---|---|
| | | | LUT | reg. | cycles@period(ns) | |
| Ex. 3 | Ours | 2 | 2,944 | 0 | 1@67.0 | $2^{-24}$ |
| | | 4 | 2,742 | 0 | 1@35.1 | |
| | | 8 | 2,582 | 0 | 1@33.1 | |
| | | 16 | 2,856 | 0 | 1@31.2 | |
| | | | 1,565* | 0 | 1@29.0* | |
| | FloPoCo | – | 3,622 | 0 | 1@55.7 | |
| Ex. 4 | Ours | 2 | 19,564 | 0 | 1@139.6 | $2^{-48}$ |
| | | 4 | 23,052 | 0 | 1@92.5 | |
| | | | 21,179* | 0 | 1@131.5* | |
| | | 8 | 15,388* | 0 | 1@250.7* | |
| | | 16 | 12,878* | 0 | 1@76.9* | |
| | | 32 | 3,909* | 0 | 1@86.7* | |
| | FloPoCo | – | 20,494 | 0 | 1@139.9 | |
| Ex. 5 | Ours | 2 | 19,423 | 0 | 1@368.1 | $2^{-48}$ |
| | | 4 | 13,642 | 0 | 1@70.3 | |
| | | 8 | 18,653 | 0 | 1@58.6 | |
| | FloPoCo | – | – | – | – | |

# Our work

- efficient methods for (quasi)optimal E-fraction approximation
- FPGA-optimized implementation of the E-method
  - efficient HW implementation of rational functions
  - customizable pipelined design $\rightarrow$ high throughput
- automatic open source tool for function evaluation
  - written in C++
  - available at: https://github.com/sfilip/emethod

**Polynomial or rational function?**

- depends on the problem!

**Thank you!**

Scan me!