Return of the hardware floating-point elementary functions

Jérémie Detrey, Florent de Dinechin, and Xavier Pujol

Projet Arénaire – LIP
UMR CNRS – ENS Lyon – UCB Lyon – INRIA 5668
http://www.ens-lyon.fr/LIP/Arenaire/
Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion
Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion
A long time ago...
(in a galaxy not so far away)

▶ a bit of paleo-bibliography
A long time ago...
(in a galaxy not so far away)

▶ a bit of paleo-bibliography

- M. D. Ercegovac (IEEE TC, 1975)
  *Radix-16 evaluation of certain elementary functions.*

  *Should the elementary functions be incorporated into computer instruction sets?*

- C. Wrathall and T. C. Chen. (ARITH 4, 1978)
  *Convergence guarantee and improvements for a hardware exponential and logarithm evaluation scheme.*

- P. Farmwald (ARITH 5, 1981)
  *High-bandwidth evaluation of elementary functions.*

- M. Cosnard, A. Guyot, B. Hochet, J.-M. Muller, H. Ouaouicha, P. Paul, and E. Zysmann (ARITH 8, 1987)
  *The FELIN arithmetic coprocessor chip.*
FPUs strike back

... then came the floating-point unit

- dedicated efficient hardware operators
- only basic operations: $+, -, \times, \div$ and $\sqrt{}$
FPUs strike back

... then came the floating-point unit

- dedicated efficient hardware operators
- only basic operations: $+$, $-$, $\times$, $\div$ and $\sqrt{\cdot}$

what about elementary functions?

- comparatively rare operations
- hardware implementation would be a waste of silicon
- dedicate silicon to more useful units (ALUs, FPUs, caches)
FPUs strike back

... then came the floating-point unit

- dedicated efficient hardware operators
- only basic operations: $+, -, \times, \div$ and $\sqrt{}$

what about elementary functions?

- comparatively rare operations
- hardware implementation would be a waste of silicon
- dedicate silicon to more useful units (ALUs, FPUs, caches)

only software or micro-code implementations
FPGAs: a new hope?

- Field-Programmable Gate Arrays
- reconfigurable integrated circuits
FPGAs: a new hope?

- Field-Programmable Gate Arrays
- reconfigurable integrated circuits
- architecture based on programmable logic cells and routing resources
  - lower performances than ASICs
  - high flexibility
  - fine-grain parallelism
  - lower cost per unit
FPGAs: a new hope?

- **Field-Programmable Gate Arrays**
- **reconfigurable** integrated circuits
- architecture based on **programmable logic cells** and **routing resources**
  - lower performances than ASICs
  - high flexibility
  - fine-grain parallelism
  - lower cost per unit

- **1 billion transistor FPGAs: huge computational capacity**

- **many application domains:**
  - digital signal and image processing
  - cryptography
  - bioinformatics
  - scientific computing
  - ...
FPGAs and arithmetic

- initially: LUT-based logic cells
FPGAs and arithmetic

- initially: LUT-based logic cells
- currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18 × 18 bits)
  - multiply-and-accumulate blocks
- not enough for many applications
FPGAs and arithmetic

- Initially: LUT-based logic cells

- Currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18 × 18 bits)
  - multiply-and-accumulate blocks

- Not enough for many applications

- Strong need for more complex operators
  - Other operations: division, square root, elementary functions, ...
  - Other number systems: modular arithmetic, real arithmetic, ...
FPGAs and arithmetic

- Initially: LUT-based logic cells
- Currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18 × 18 bits)
  - multiply-and-accumulate blocks
- Not enough for many applications
- Strong need for more complex operators
  - Other operations: division, square root, elementary functions, ...
  - Other number systems: modular arithmetic, real arithmetic, ...
FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision
**FPLibrary**

- library of **portable VHDL operators for floating-point**

- all operators are **parameterized in terms of range and precision**

<table>
<thead>
<tr>
<th></th>
<th>single precision</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/−</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>÷</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>√</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
FP Library

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

<table>
<thead>
<tr>
<th></th>
<th>single precision</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+/-)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(\times)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(\div)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(\sqrt{\cdot})</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(\log x)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>(e^x)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>(\sin x / \cos x)</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
FPLibrary

- library of portable VHDL operators for floating-point

- all operators are parameterized in terms of range and precision

<table>
<thead>
<tr>
<th></th>
<th>single precision</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+/-$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\times$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\div$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\sqrt{}$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\log x$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$e^x$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\sin x$ / $\cos x$</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- single-precision logarithm and exponential
  - hardware-specific algorithms
  - ad-hoc range reduction
  - table-based fixed-point evaluation
  - small and fast operators
FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

<table>
<thead>
<tr>
<th></th>
<th>single precision</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+/-) (\times) (\div) (\sqrt{}) (\log x) (e^x) (\sin x / \cos x)</td>
<td>✓✓ ✓✓ ✓✓ ✓✓ ✓✓ ✓✓</td>
<td>✓✓ ✓✓ ✓✓ ✓✓ ✓✓ ✓✓</td>
</tr>
</tbody>
</table>

- single-precision logarithm and exponential
  - hardware-specific algorithms
  - ad-hoc range reduction
  - table-based fixed-point evaluation
  - small and fast operators
Double precision: using the same method?

- range reduction and reconstruction are scalable
Double precision: using the same method?

- **range reduction and reconstruction** are scalable

- **table-based method** for the actual computation
  - exponential growth of the area
  - estimations w.r.t. single precision: $15 \times$ larger for the exponential, and $40 \times$ larger for the logarithm!!
  - unacceptable overhead for usual FPGAs

- need for another algorithm, suited to higher precisions
Double precision: using the same method?

- Range reduction and reconstruction are scalable

- Table-based method for the actual computation
  - Exponential growth of the area
  - Estimations w.r.t. single precision: \(15 \times\) larger for the exponential, and \(40 \times\) larger for the logarithm!!
  - Unacceptable overhead for usual FPGAs

- Need for another algorithm, suited to higher precisions

- Iterative method
  - Smaller architecture
  - Higher scalability
  - Longer critical path
Outline of the talk

▶ Context

▶ Double-precision exponential

▶ Results

▶ Conclusion
Number format

- 2 parameters: \( w_E \) (range) and \( w_F \) (precision)

- inspired from the IEEE-754 standard:

\[
X = (-1)^{S_X} \cdot 1.F_X \cdot 2^{E_X - E_0}
\]
Number format

- 2 parameters: $w_E$ (range) and $w_F$ (precision)

- inspired from the IEEE-754 standard:

$$X = (-1)^{S_X} \cdot 1.F_X \cdot 2^{E_X - E_0}$$

- 2 extra bits for exceptional cases: zero, infinity or Not-a-Number (NaN)
Evaluation method

▶ range reduction:

\[ X = k \cdot \log 2 + Y \quad \text{with } k \in \mathbb{Z} \text{ and } 0 \leq Y < 1 \]

▶ we obtain:

\[ R = e^X = 2^k \cdot e^Y \]
Evaluation method

- range reduction:

\[ X = k \cdot \log 2 + Y \quad \text{with} \quad k \in \mathbb{Z} \quad \text{and} \quad 0 \leq Y < 1 \]

- we obtain:

\[ R = e^X = 2^k \cdot e^Y \]

- fixed-point \( e^Y \)?
Evaluation method

- range reduction:

\[ X = k \cdot \log 2 + Y \quad \text{with } k \in \mathbb{Z} \text{ and } 0 \leq Y < 1 \]

we obtain:

\[ R = e^X = 2^k \cdot e^Y \]

- fixed-point \( e^Y \)?
  generalization of an idea by Wong and Goto (IEEE TC 1994)

  - successive range reductions of the fixed-point argument \( Y \)
  - once the argument sufficiently reduced, direct evaluation of the exponential
  - reconstructions using rectangular multipliers
  - computes \( e^Y - 1 \)
Iterative method: range reductions

- for step each $i$, we consider the argument $Y_i$ (starting with $Y_0 = Y$)
Iterative method: range reductions

- for step each $i$, we consider the argument $Y_i$ (starting with $Y_0 = Y$)

\[ \pm 0.0 \ldots 0 \]
\[ A_i \quad B_i \]

- splitting $Y_i$ as $A_i + B_i$, we address two look-up tables with $A_i$:
  - $e^{A_i} - 1$, rounded to its $\alpha_i$ most significant bits, noted $\tilde{e}^{A_i} - 1$
  - $L_i = \log (\tilde{e}^{A_i})$, rounded to its $\alpha_i + \beta_i$ most significant bits
Iterative method: range reductions

- for step each $i$, we consider the argument $Y_i$ (starting with $Y_0 = Y$)

  \[ Y_i = A_i + B_i \]

  - splitting $Y_i$ as $A_i + B_i$, we address two look-up tables with $A_i$:
    - $e^{A_i} - 1$, rounded to its $\alpha_i$ most significant bits, noted $\tilde{e}^{A_i} - 1$
    - $L_i = \log(\tilde{e}^{A_i})$, rounded to its $\alpha_i + \beta_i$ most significant bits

  - by construction, $L_i \approx Y_i$
Iterative method: range reductions

for step each \( i \), we consider the argument \( Y_i \) (starting with \( Y_0 = Y \))

\[
\begin{array}{c|c|c}
\pm 0.0 & A_i & \pm 0.0 \\
\hline
\pm 0.0 & L_i & \pm 0.0 \\
\hline
\pm 0.0 & Y_{i+1} & \pm 0.0 \\
\end{array}
\]

splitting \( Y_i \) as \( A_i + B_i \), we address two look-up tables with \( A_i \):

- \( e^{A_i} - 1 \), rounded to its \( \alpha_i \) most significant bits, noted \( \tilde{e}^{A_i} - 1 \)
- \( L_i = \log(\tilde{e}^{A_i}) \), rounded to its \( \alpha_i + \beta_i \) most significant bits

by construction, \( L_i \approx Y_i \)

we then define \( Y_{i+1} \) as \( Y_i - L_i \):

- the \( \alpha_i - 1 \) most significant bits of \( Y_i \) are cancelled
- \( Y_{i+1} \) is a \( 1 + \beta_i \)-bit number
Iterative method: computing the exponential

- the reduction process is iterated until the step \( k \) such that

\[
Y_k < 2^{-\left\lfloor w_F/2 \right\rfloor}
\]
Iterative method: computing the exponential

- the reduction process is iterated until the step $k$ such that

$$Y_k < 2^{-\lceil \frac{w_F}{2} \rceil}$$

- we can then approximate the exponential as

$$e^{Y_k} - 1 \approx Y_k$$
Iterative method: reconstructions

- at each step $i$, we have:
  - $\tilde{e}^{A_i} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log \left( e^{A_i} \right)$
Iterative method: reconstructions

At each step $i$, we have:

- $\tilde{e}^A_i - 1$, from the corresponding range reduction step
- $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log(\tilde{e}^A_i)$

We then compute $e^{Y_i} - 1$ as

$$
\left(\tilde{e}^A_i - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\tilde{e}^A_i - 1\right) + \left(e^{Y_{i+1}} - 1\right)
$$
Iterative method: reconstructions

- at each step $i$, we have:
  - $\tilde{e}^A_i - 1$, from the corresponding range reduction step
  - $e^{Y_i+1} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log(\tilde{e}^A_i)$

- we then compute $e^{Y_i} - 1$ as

\[
\left(\tilde{e}^A_i - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\tilde{e}^A_i - 1\right) + \left(e^{Y_{i+1}} - 1\right) \\
= \tilde{e}^A_i \cdot e^{Y_{i+1}} - 1
\]
Iterative method: reconstructions

- at each step $i$, we have:
  - $\tilde{e}^{A_i} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log(\tilde{e}^{A_i})$

- we then compute $e^{Y_i} - 1$ as

\[
\left(\tilde{e}^{A_i} - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\tilde{e}^{A_i} - 1\right) + \left(e^{Y_{i+1}} - 1\right)
= \tilde{e}^{A_i} \cdot e^{Y_{i+1}} - 1
= \tilde{e}^{A_i} \cdot e^{Y_i} \cdot e^{-\log(\tilde{e}^{A_i})} - 1
\]
Architecture

\[ \text{sign / exception handling} \]

\[ \text{normalize / round} \]

\[ e^Y - 1 \]

\[ +1 \]

\[ e^Y \]

\[ E_0 \]

\[ + \]

\[ - \]

\[ \times \]

\[ \log 2 \]

\[ 1/\log 2 \]

\[ \times \]

\[ \pm1 \]

\[ \text{shift} \]

\[ \text{overflow/underflow} \]

\[ X_{\text{fix}} \]

\[ k \]

\[ \text{round} \]

\[ \text{exn} X \]

\[ S_X \]

\[ E_X \]

\[ F_X \]

\[ \tilde{R} \approx e^X \]
Architecture

\[ e^Y - 1 \]

Sign / exception handling

Normalize / round

\[ \tilde{R} \approx e^X \]

J. Detrey, F. de Dinechin, and X. Pujol – Return of the hardware floating-point elementary functions
Architecture

\[ E_0 \]

1/ log 2

\[ \times \pm 1 \]

shift

overflow, underflow

\[ e^{A_0} - 1 \]

\[ \log (e^{A_0}) \]

- 

\[ Y_1 \]

\[ \bar{R} \approx e^X \]

J. Detrey, F. de Dinechin, and X. Pujol – Return of the hardware floating-point elementary functions
Architecture

\[ \tilde{R} \approx e^X \]

\[ e^Y - 1 \]

\[ \log (e^A) \]

\[ e^A_0 - 1 \]

\[ \frac{1}{\log 2} \]

\[ \pm 1 \]

\[ \text{shift} \]

\[ \text{overflow, underflow} \]

\[ \text{normalize / round} \]

\[ \text{sign / exception handling} \]
Architecture

\[ e^Y - 1 \]
Architecture

\[
\begin{align*}
\tilde{R} & \approx e^X \\
E_0 & + 1 \quad \text{normalize / round} \\
E_0 & + \text{sign / exception handling} \\
E_0 & \log 2 \\
\times & e^{A_0 - 1} \\
\times & \log (e^{A_0}) \\
\times & - \\
\times & e^{Y_1 - 1} \\
\times & e^{Y_k - 1} \\
\times & e^{A_0} - 1 \\
\times & e^{A_0} - 1 \\
\times & \text{shift} \\
\times & \text{overflow / underflow} \\
1 & \log 2 \\
1 & \text{round} \\
E_0 & \pm 1 \\
Y & \text{Xfix} \\
E_0 & - \\
E_0 & - 1 \\
Y & \text{Y} \\
A_0 & B_0 \\
A_1 & B_1 \\
Y_k & - \\
Y & - \\
\end{align*}
\]
Architecture

\[ \log_2(X) \pm 1 \]

\[ \log_2(Y) \]

\[ e^Y - 1 \]

\[ \tilde{R} \approx e^X \]

\[ A_0 \]

\[ B_0 \]

\[ A_1 \]

\[ B_1 \]

\[ e^{A_0} - 1 \]

\[ \log(e^{A_0}) \]

\[ e^{A_0} - 1 \]

\[ e^{Y_1} - 1 \]

\[ e^{Y_0} - 1 \]

\[ e^{Y_k} - 1 \]
Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion
single precision \((w_E, w_F) = (8, 23)\) (table-based method):
938 slices (18% of a Virtex-II 1000 FPGA)
single precision \((w_E, w_F) = (8, 23)\) (table-based method):
938 slices (18% of a Virtex-II 1000 FPGA)
single precision \((w_E, w_F) = (8, 23)\) (table-based method):
938 slices \((18\% \text{ of a Virtex-II 1000 FPGA})\)

double precision \((w_E, w_F) = (11, 52)\) (iterative method):
2045 slices \((40\%)\)
Operator latency (exponential)

<table>
<thead>
<tr>
<th>Precision $w_F$ (in bits)</th>
<th>Latency (in ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>110</td>
</tr>
<tr>
<td>18</td>
<td>120</td>
</tr>
<tr>
<td>22</td>
<td>130</td>
</tr>
<tr>
<td>26</td>
<td>140</td>
</tr>
<tr>
<td>30</td>
<td>150</td>
</tr>
<tr>
<td>34</td>
<td>160</td>
</tr>
<tr>
<td>38</td>
<td>170</td>
</tr>
<tr>
<td>42</td>
<td>180</td>
</tr>
<tr>
<td>46</td>
<td>190</td>
</tr>
<tr>
<td>50</td>
<td>200</td>
</tr>
</tbody>
</table>

- **single precision** $(w_E, w_F) = (8, 23)$ (table-based method): 97 ns
- **double precision** $(w_E, w_F) = (11, 52)$ (iterative method): 229 ns
Outline of the talk

▶ Context
▶ Double-precision exponential
▶ Results
▶ Conclusion
Our contribution

- exponential and logarithm operators
- up to double precision
- guaranteed faithful rounding
- scalable method
- hardware-specific algorithms: fast and cheap operators
Future work

- pipeline
- implement **double precision** for other functions for **FPLibrary**
- study **compound functions**
Future work

▶ pipeline

▶ implement double precision for other functions for FPLibrary

▶ study compound functions

▶ careful error analysis:
  - certified algorithms and operators
  - generic proofs (Gappa)

▶ most of this work is not FPGA-specific: extend it to ASICs
Thank you for your attention

► more information & download page:
http://www.ens-lyon.fr/LIP/Arenaire/
Thank you for your attention

more information & download page:
http://www.ens-lyon.fr/LIP/Arenaire/

Questions?